Ontology-based Semantic Query Processing in Database Systems

DISSERTATION

zur Erlangung des akademischen Grades
doctor rerum naturalium
(Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
Herr Dipl.-Inf. Ben Necib Chokri
geboren am 07.01.1970 in Tunesien

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Christoph Markschies

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Wolfgang Coy

Gutachter:

1. Prof. Johann-Christoph Freytag, Ph.D.
2. Prof. Dr. Felix Naumann
3. Prof. Dr. Kai-Uwe Sattler

eingereicht am:	11. Dezember 2006
Tag der mündlichen Prüfung:	30. November 2007

Abstract

Currently, database management systems solely rely on exact syntax of queries to retrieve data. As consequence query answers often do not meet the user's intention. In this thesis we propose an ontology-based semantic query processing approach for database systems. We use ontologies to transform a user query into another query that may provide a more meaningful answer to the user. For this purpose, we define and specify different mappings that relate concepts of an ontology with those of an underlying database and develop a set of algorithms that allow us to find these mappings in a semi-automatic way. Moreover, we propose a set of semantic rules for transforming queries using terms derived from the ontology. We classify the rules and demonstrate their usefulness using practical examples. Furthermore, we make use of the theory of term rewriting systems to formalize the transformation process and to study the basic properties for applying these rules. Finally, we implement a prototype system using current technologies and evaluate its capability by using a real world application.

Keywords:
Databases, Query Processing, Ontologies, Semantic

Zusammenfassung

Die Bedeutung der in den relationalen Datenbankmanagementsystemen dargestellten Realwelt-Objekten wird weder explizit noch vollständig beschrieben. Demzufolge treffen häufig diese Systeme mit den Anfrageantworten nicht die Benutzerabsichten. Die vorliegende Dissertation präsentiert einen ontologie-basierten Ansatz für die semantische Anfrageverarbeitung. In diesem Ansatz sollen semantische Informationen aus einer gegebenen Ontologie abgeleitet und für die Umformulierung der Benutzeranfrage verwendet werden. Dies führt zu einer neuen Anfrage, die für den Benutzer sinnvollere Ergebnisse aus der Datenbank zurückliefern kann. Wir definieren und spezifizieren Einschränkungen und Abbildungen zwischen der Ontologie- und den Datenbank-Konzepten, um eine Ontologie mit einer Datenbank zu verknüpfen. Des Weiteren entwickeln wir eine Reihe von Algorithmen, die uns helfen, diese Abbildungen auf eine halbautomatische Weise zu finden. Außerdem entwickeln wir eine Reihe von semantischen Regeln, die für die Umformulierung einer Anfrage benutzt werden. Die Haupteigenschaft einer Regel ist es, Begriffe einer Anfrage durch andere Begriffe zu ersetzen oder anzureichern, die von denselben ontologischen Konzepten dargestellt werden. Weiterhin benutzen wir die Theorie der Termersetzungssysteme, um den Transformationsprozess zu formalisieren und die wesentlichen Eigenschaften für das Anwenden der Regeln zu studieren. Aufbauend auf diesem Ansatz wurde ein Prototyp implementiert und wurde die Fähigkeit unseres Ansatzes durch einer real existierenden Anwendung ausgewertet.

Schlagwörter:
Datenbanken, Anfragebearbeitung, Ontologien, Semantik

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In physics, bioinformatics, geography, and other disciplines, we encounter many applications that both access and store large data sets in databases. Usually, data is collected from different information sources or generated by different users. Hence, they could be interpreted differently. In this context, query answering must be efficient. During the past decades, relational database management systems (RDBMS) have proven to be a technology that is able to deal with complex queries over large data sets. To retrieve data from a database, users submit queries using terms that represent some real world objects and expect as result all database items that represent those objects. The queries are usually specified by imposing conditions on the attribute values stored in the database. However, the answers to these queries might not meet the user intension. Often, users must reformulate their queries using different values until they obtain a satisfactory answer. One of the reasons is that existing DBMSs only rely on query syntax, i.e., they retrieve only data that exactly match query expressions.

In this context, traditional approaches to database management have not paid much attention to the semantics of user queries, i.e., whether they return results meeting the user's intention, rather they focus on providing support for efficient query processing. In query optimization efficiency means the ability of the system to minimize the costs of query execution. There, the main goal is to rewrite a given query into another equivalent one that uses less time and/or resources. Two queries are considered to be equivalent if they return the same result.

While there is much work being done on improving the efficiency of query processing, not much has been published on improving the quality of query answers. Recently, some approaches attempt to extend conventional database systems to use similarity metrics in query processing [AGG02, BCM02, Sri03, DHL+04]. The metrics are either provided by the user or computed by the system. However, these approaches require to modify the existing query languages by redefining their operators. Further, estimating similarity between terms of a query and values in the database is very difficult. These difficulties arise due to many reasons such as the unfamiliarity of users with the database content, the difficulty in assigning weights, or different priorities. As a consequence, the system might return a large number of tuples to the user and leaves to him the task to select relevant results. In addition, the system requires to scan the entire database for computing similarities leading to poor performance [Bin02].

However, there is a real need for approaches that can improve query answers. An important approach is the use of "semantic knowledge" about the database and its content to semantically improve query processing. Recent work uses ontologies as the source of such knowledge. Users formulate their queries over an ontology without knowing where and how the data is organized in the database. However, due to their large size, navigating through ontology and selecting appropriate terms for the queries causes many problems. In addition, users might not be familiar with the ontology content. Finding the suitable data source for answering queries is a problem addressed extensively in data integration systems [Qua05] and is out of the scope of this thesis. We assume that users know about the structure of the database but not about its content.

The main goal of this work is to develop a new approach to improve the effectiveness of answering queries based on ontologies. A user query might be transformed into a new query which is not necessary equivalent, but might provide a more meaningful answer satisfying the user's intention. "Meaningful answer" means that that it is more complete than the initial one w.r.t. user's intension. To achieve this goal, we must answer some fundamental questions:

1. How can semantics be captured, expressed, and used?

   Database schema and integrity constraints do not provide enough semantics to describe the meaning of real world entities represented in a database. We need an additional source that can provide more semantics, and hence features and aspects of the entities. The source must be able to explicitly express and reason about semantics.

2. How can we establish a mapping between an ontology and a database?

   Our ultimate goal is to improve query answers using semantic information from ontologies. To achieve this goal, we have to establish a mapping between database concepts with associated ontology components. This mapping must be specified sufficiently such that most of the relevant results could be retrieved.

3. How can ontologies be used to improve query answering?

   We need a generic and flexible approach which is independent of the ontology content. We aim at developing a set of semantic rules which can extract semantics from any ontology. The rules will be used for the query transformation. They should depend only on the mapping information about the ontology and the corresponding database. We opted for an approach that can be implemented without substantial changes to the components of a database system and hence could be easily implemented in many existing systems. Users formulate their queries over the database as usual and the system extracts semantics and transforms the queries.

In this thesis we will discuss these questions and thereby develop new concepts and methods for query processing in a semantical meaningful way.

This thesis is organized as follows. Chapter 2 gives an overview of query processing in a relational database system and presents a brief survey of query processing literature, with a focus on query rewriting techniques that utilize semantics.

Chapter 3 presents the basic notions and ideas used for this thesis. It starts by providing clear definitions of semantics and their related issues such as syntax, models, and conceptual schema. Then, it proposes ontologies as a main support for a semantic query transformation. A clear definition is given of what is meant by an ontology in this thesis. A discussion of the potential role of ontologies in describing the database content is also presented.

Chapter 4 is concerned with the use of ontologies in query processing. It explains how ontologies help to resolve semantic problems and how they can act as a query model. Furthermore, this chapter specifies the mapping between a database and an ontology at the instance level as well as at the schema level. We propose a set of algorithms that allows us to find concepts of the database and the ontology that are related. We also review and compare some prototype systems related to our work. We focus on how they apply ontologies to query processing.

In Chapter 5, we present a new approach to semantic query processing within single (object) relational database systems. We develop a set of transformation rules and illustrate how they can be effectively used for improving query answers. We classify the rules into two subsets: rules that extend the original query results and rules that reduce them. In both cases the results better satisfy the user's need and better meet his intention. We outline important concepts of the theory of term rewriting systems and use them to study the main properties of the rules application.

Chapter 6 describes the design and the implementation of a prototype based on concepts and methods presented in Chapters 4 and 5. We investigate the standard technologies that can be used for the implementation and develop an approach to represent mapping information. We also illustrate technical details about the main components of the prototype architecture. We evaluate the effectiveness of our semantic transformation by implementing a real world application and executing a set of test queries.

Finally, Chapter 7 summarizes the results of this thesis and outlines future work that may extend and further improve them.

Chapter 2

Query Processing in Databases

Over the past decade, the use of database systems has been dramatically expanded with many different applications ranging from traditional applications (e.g., bank transactions, airline reservation) to more advanced applications (e.g., multimedia storage and analysis, internet services). Database management systems have become successful because of their ability to store and retrieve large data efficiently. Thereby, query processing represents a challenge for database systems because user queries are expected to be executed as fast as possible and with as little computing costs as possible. Moreover, database systems have to return answers that will best meet the user's intentions. A wide variety of work has addressed many problems related to query processing in various database environments (centralized, distributed, and mobile environments) including query optimization [FMV93, JMP97, SJS01, MD03], data indexing [FL01, HAI02, ZAF+03] and query language extensions [ART96, LS03, LWZ04].

This chapter reviews the theoretical foundations relevant to query processing in relational database systems. Our objective is not to give a full coverage of all concepts and techniques used for query processing but to provide the terminologies used in the subsequent chapters and to illustrate some of the techniques that are most relevant to our work. The reasons of choosing the relational framework are numerous. First, relational databases are based on mathematical foundations making easy to treat complex problems at the conceptual level. Second, they have proved to be adequate for extending their concepts and for integrating new approaches [FMV93]. Third, relational database systems use high-level languages which give many opportunities for optimization. Finally, relational database systems are broadly deployed in many applications and information services in the commercial market.

Moreover, we focus our discussion on query processing techniques developed for centralized databases because of the relevance to our approach. Nevertheless, most of these techniques are reused and adapted to other kinds of database systems [FMV93, Gru93, Kos00].

This chapter is organized as follows. We start by defining standard concepts from the database theory used in this thesis. We then give an overview of the basic steps of query processing. Since transformation of queries is the basis of our approach we illustrate the phases where the transformations occurs and survey the most relevant approaches.

2.1 Database Systems

A *database* as defined by Elmasri et al. [NE01] is a collection of related data which has the following specific properties:

- A database represents a part of the real world, called the *universe of discourse* (UoD).
- A database is designed for some applications concerned with that universe.
- A database organizes data into a logical coherent structure, called a *data model*.

The system that allows users to create, maintain, and request underlying database(s) is called *database management system* (DBMS). Furthermore, a *database system* denotes a combination of a DBMS with its database(s).

2.1.1 Relational Database Systems

Relational database technology is the de-facto standard for current data management systems. One of the keys for their success is the data model, the *relational model*, due to many powerful features such as:

- It is a simple mathematical model. Data is structured as relations (tables).
- It provides a clear separation between the logical and physical level. This implies a flexibility for defining database objects related to data as well as the operations.
- The manipulation of relations is non-procedural (*declarative*), i.e., database designer and users do not have to explicitly specify how to access or to retrieve data from the database. This task is performed by the DBMS.

Terminologies. In formal terms, a relational database is a collection of associated relations. Each relation has a name and a set of *attributes*. Let $U = \{A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n\}$ be a universe of attribute names where to each $A_i \in U$ corresponds a finite set D_i called *domain* of A_i. Let $D = D_1 \cup D_2 \ldots D_n$. A database relation R is a named object which consists of a schema and a body (extension). A *relation schema* $R(A_1, A_2, \ldots, A_m)$ is a finite subset of U. The number m denotes the *arty* of R. A *relation instance* (a *tuple*) of R is a function μ from R to D with the restriction that $\mu(A_i) \in D_i$. A *relation* on R, denoted $r(R)$, is a finite set of instances $\{\mu_1, \ldots, \mu_p\}$. Thus, a relation is a subset of the cartesian product $D^m = D_1 \times D_2 \ldots \times D_m$. A *relation extension* (extent), denoted $Ext(R)$, is the set of all instances on R (also called base relation). A *database schema* is defined as a pair $S = (\Sigma, I)$ where Σ is a finite set of relation schemas $\{R_1, \ldots, R_q\}$ and I is a set of predicates specifying the *integrity constraints* that must hold in the database. Constraints are properties that data must satisfy at all times. A *database extension* on Σ, denoted $Ext(DB)$, is a set of relation extensions $\{Ext(R_1), \ldots, Ext(R_q)\}$ where $R_i \in \Sigma$. A tuple from $Ext(DB)$ is also called a database instance. Given $x \in D$, x is called a *database value* if there exists a tuple

$\mu \in Ext(DB)$ such that $x = \mu(A)$ where $A \in \Sigma$ [Tha93]. We also define a *derived relation* as being a relation that is defined in terms of other (base) relations, i.e., it is obtained by computing a query expression. A derived relation is also called a *virtual view*.

In summary, a relational database DB is defined as $DB = \{S(\Sigma, I), Ext(DB)\}$.

Assumptions. For this thesis we make some assumptions about the design of the database schema:

- The relational schemas are at least in third normal form (3 NF) [Tha93].
- All attributes in the database schema have different names.
- Primary keys are single "artificial" attributes (extra attributes which do not exist in the external reality).
- Integrity constraints are given only in form of key constraints (primary and foreign keys).

Example 2.1 (University Database) We use the following university database in our examples throughout the thesis. The database describes employees and students in a university and has the following relational schema:

Employee(SSN, FName, LName, Position, DNr)
PKey(SSN), FKey(Dept) references Department

Student(INr, SName, Degree, Year)
PKey(INr)

Department(DNr, DName, Institute)
PKey(DNr)

Division(UNr, DNr, UName)
PKey(UNr), FKey(DNr) references Department

Course(CID, Cname)
PKey(CID)

CS-Courses(CID, Level)
PKey(CID), FKey(CID) references Course

Lecturer(SSN, CID)
PKey(SSN,CID), FKey(SSN) references Employee, FKey(CID) references Course

Enrollment(INr, CID)
PKey(INr,CID), FKey(INr) references Student, FKey(CID) references Course

We denote a primary key by *PKey(id)* where *id* is an attribute. Furthermore, "*FKey(id-name) references Rel-name*" refers to a foreign key constraint: The key *id-name* references the primary key of the relation *Rel-name*.

□

2.1.2 Queries, Query Languages

A *query* can be seen as an expression that specifies properties of the data to be retrieved. Basically, a query specifies:

- Relations that are accessed in the database;
- Conditions that must be checked;
- Aspects of the qualifying tuples that must be retrieved.

To this end, DBMSs offer a special programming language, called a *query language*. There are two fundamental classes of query languages for relational DBMSs: *relational calculus* based languages and *relational algebra* based languages. The key feature that distinguish both classes is how a user can formulate a query. Relational calculus languages are *declarative*. That is, when a user poses a query against a database he declares *what* information the answer should contain. He does not care about how this answer should be computed. However, relational algebra languages are more operational (*functional*). That is, a user must formulate his queries by indicating to the system what and *how* to obtain the answer. We note that both relational calculus and relational algebra languages are proved to be in terms of expressive power. A query is safe, if for any relation contents, it always results in a finite set of entries. Any safe relational calculus query can be translated into an equivalent relational algebra query, and vice versa [Cod72]. We assume that the reader is familiar with relational algebra and we restrict our representation of query languages to relational calculus because they will be extensively used in this thesis.

2.1.3 Relational Calculus

Relational calculus languages are classified into two groups: *Domain relational calculus* (DRC) and *tuple relational calculus* (TRC) [LP82]. The difference is in terms of variables used in formulas. While a variable denotes a tuple of a relation in TRC, it denotes a value of a domain in DRC. We briefly review the domain relational calculus.

The DRC language considers a relational database as a collection of domain values. Relational calculus formulas contain variables that are based on domains of attributes, called *domain variables*. Thus, each domain variable specifies an attribute value of a tuple. Formulas in DRC are *well-formed*. A *well-formed formula* (wff) is an expression built up from atoms and a collection of operators as follows:

Let u, v and w be domain variables, t be a constant, and θ be an arithmetic comparison operator ($=$, $\neq$, $<$, $\leq$, $>$). The atoms are of three types: $R(x_1, \ldots x_n)$, $v\theta w$, and $u\theta t$. R is a predicate representing an n-airy relation in the database. Formulas and variables can be also defined recursively using the logic operators "$\wedge$", "$\vee$, "$\Longrightarrow$", and "$\neg$" (negation) as follows:

- An atom is a formula.
- If p is a formula then $\neg p$ is also a formula.
- If p_1 and p_2 are formulas then $p_1 \wedge p_2$, $p_1 \vee p_2$ and $p_1 \Longrightarrow p_2$ are also formulas.
- If $p(x)$ is a formula and x is a free variable then $\exists\ p(x)$ and $\forall\ p(x)$ are formulas. For the simplicity, we denote formulas as $\exists\ x_1\, x_2\ p(x_1, x_2)$ instead of $\exists\ x_1(\exists\ x_2\ p(x_1, x_2))$ [KE01].

A variable is free in p if it occurs in a position where it is not bound by an enclosing existential or universal quantifier.

Definition 2.1 (DRC Query) *A query Q in DRC is defined as*

$$Q = \{(x_1, \ldots x_n) \mid p(x_1, \ldots x_n)\}$$

where x_i $(1 \leq i \leq n)$ are domain variables. p is a well-formed formula in which all x_i's free.

Example 2.2 Consider a query that retrieves from the university database the names of the students who attend courses in the second semester. In DRC it is expressed as

$$\{n \mid \exists\, i\, g\, y\ Student(i, n, g, y) \wedge \exists\, c\, r\, s\ [Course(c, r, s) \wedge \exists\ i^{'}\ c^{'}\ [Enroll(i^{'}, c^{'}) \wedge i = i^{'} \wedge c = c^{'} \wedge s = 2\,]]\}.$$ □

2.2 Query Processing

Query processing is one of the main task of a database management system. For this task, the query processor receives a user query in high-level language (e.g., DRC or SQL) as input, converts it into an internal representation, retrieves the relevant data from the database using efficient methods, and returns the output results to the user. Since a database may contain large amounts of data the objective of a query processor is performing this task faster using a minimum of the computing resources available to the DBMS. Figure 2.1 depicts the overall architecture for query processing. In this section we describe the basic steps of query processing focusing on query transformation, and present the different types of query optimization. For the sake of simplicity we present each step independently.

2.2.1 Basic Steps of Query Processing

Following [Dat02] we describe the different steps of query processing in (centralized) DBMSs as follows.

1. Parse the query input and convert it into an internal representation.
2. Transform the query into a standardized canonical form, simplify it by eliminating redundancy and ameliorate it to possible equivalent queries.
3. Generate a query plan for each transformed query using information from the database catalog.

4. Estimate computation costs of each access plan based on physical storage of data and select the best plan.

5. Construct the execution plan by specifying the evaluation methods used for each operation in the plan.

6. Execute the resulting plan thus retrieving the resulting tuples from the database.

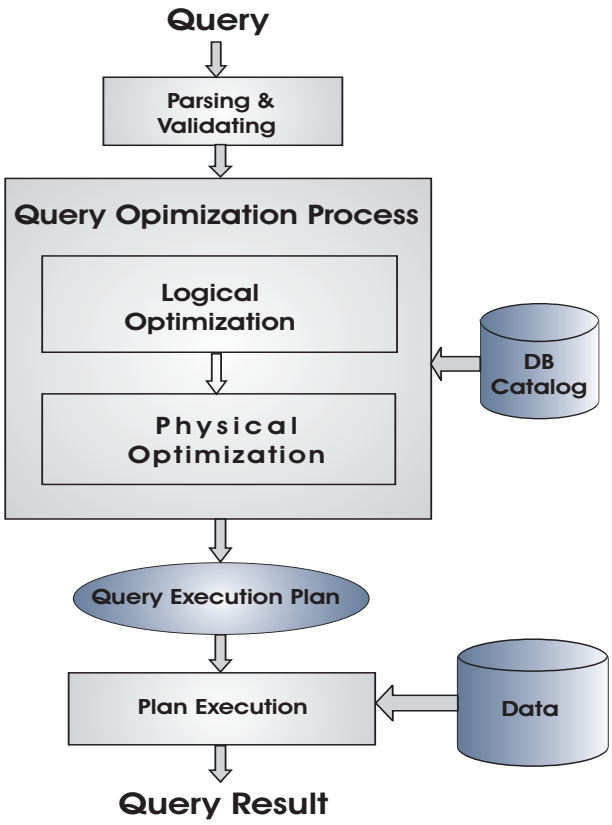


Figure 2.1: Typical architecture for Query Processing

More specifically, during the first step the query processor checks whether the query syntax is correct, e.g., the processor verifies the syntax of each operation, the existence, and the type of database elements used in the query. Then, it converts the query into a relational algebra query represented in an internal form. This representation will be suitable for performing optimizations. A typical representation for relational DBMSs is the *algebraic tree*, also called *operator-tree*. Other representations include the *Query Graph Model* as used in STARBURST system [HFLP89]. An algebraic tree provides an easier understanding of the structural properties of the operators appearing in the query expressions. The operators are represented by inner nodes in the tree and the data (flowing up from the leaves to the root) are represented by directed edges. Incoming edges represent the inputs of an operator while outgoing edges represent the outputs. Unary operators (e.g., Π or σ) have only one incoming edge while binary operators (e.g., $\times$, $\bowtie$ or $\cup$ or $\cap$) have two incoming edges. However, both types of operators produce

one output, hence they have one outgoing edge. The leaves of the tree represent the relations used in the query. The construction of an algebraic tree is achieved as follows. First, a leaf is created for each tuple variable corresponding to a relation. In SQL, the leaves are immediately available in the FROM clause. Second, the root node is created as a project operation involving the result attributes. These are available in the SELECT clause. Third, the qualification (WHERE clause) is translated into an appropriate sequence of relational operators going from the leaves to the root [OV91].

Example 2.3 Figure 2.2 depicts an algebraic tree of a query that requests the university database for students that attend courses of the professor 'Been'. The SQL-form of the query is shown on the left-hand side of the tree.

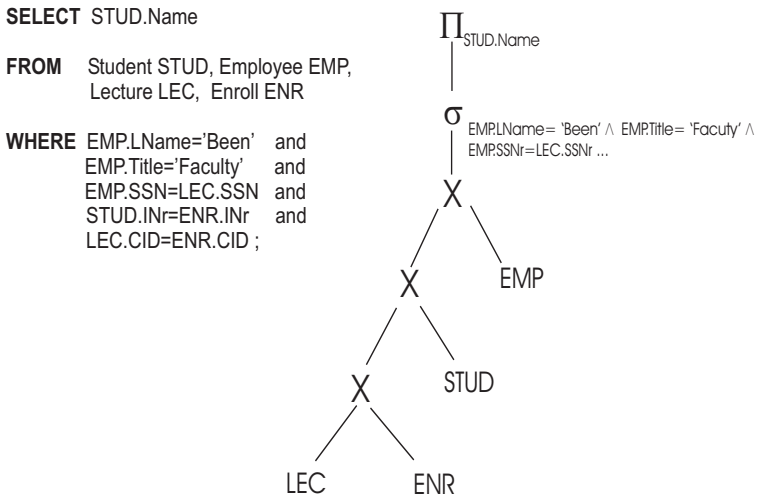


Figure 2.2: An algebraic Tree

□

Once the query is translated into an internal form, the next step transforms it into a *canonical form* to facilitate further processing. The most important transformation is that of the query qualification (selection condition in RA) which may be an arbitrary complex quantifier-free predicate preceded by universal or existential quantifiers [OV91]. There are two kinds of normal forms: *Conjunctive normal form* (CNF) and *disjunctive normal form* (DNF). On one hand the CNF is a conjunction ($\wedge$) of disjunction ($\vee$) of predicates, which has the form

$(p_{11} \vee p_{12} \ldots \vee p_{1n}) \wedge \ldots \wedge (p_{m1} \vee p_{12} \ldots \vee p_{mn})$ where p_{ij} are predicates.

On the other hand, the DNF is a disjunction of conjunction of predicates, which has the form

$(p_{11} \wedge p_{12} \ldots \wedge p_{1n}) \vee \ldots \vee (p_{m1} \wedge p_{12} \ldots \wedge p_{mn})$ where p_{ij} are predicates.

The transformation of quantifier-free predicate is simply done by moving quantifiers over terms from right to left using equivalence rules for logical operators $\wedge$, $\vee$, and $\neg$ such as the well-known idempotency rules and DeMorgen's rules [JK84].

Example 2.4 Consider the following SQL-query defined over the university database schema:
SELECT Salary
FROM Employee
WHERE (Title= 'Faculty' and (Not Title='Faculty' or Not Title='Technical')) and
Title='Technical' and Dept= 'CS.'
Using the logical rules the query can be simplified into this equivalent form:

SELECT Salary FROM Employee WHERE Title= 'Faculty' and Dept='CS.' □

The second query processing step performs only syntactical modifications. For example, it does not consider any performance parameter (e.g., the computation time, the size of intermediate results and the number of relation elements accessed) for the query execution. However, a query could be further transformed to another query that may produce the same answer but may reduce resource consumption and/or decrease response time. This step is called *query optimization*. We distinguish between two phases of query optimizations: *logical optimization* and *physical optimization*. The logical optimization precedes the physical one. These techniques will be described in the next sections in more details.

The outcome of logical transformations leads to a set of algebra expressions which are logical equivalent to the original one but could be executed with minimal costs. These expressions are represented in an algebraic form. Each query expression must then be translated into an *access plan* for further optimization. An access plan is a sequence of operations leading from the existing data structures to the query result. This is done by mapping logical operations into physical ones. Once the plan generator creates the set of logical plans, physical optimization will be performed. In this phase, each plan will be examined using cost-estimations models based on the physical storage of the data and the best plan will be selected. We note that the words "better" and "best" should not be understood literally since the algorithms that guide the optimization process are heuristic. These techniques will be discussed in more details in Section 2.5. To execute a query, the DBMS generates a detailed plan for the best plan, called *query execution plan* (QEP). A QEP can be viewed as a sequence of operations which captures the concrete evaluation method (e.g., access path) related to each operation. This plan will be then passed to the execution engine, which is a software responsible directly to generate relevant code for retrieving query results.

2.2.2 Query Rewriting for Optimization

Query optimization aims at transforming a query into an equivalent query that may be more amenable to evaluation by the execution engine. That is, the transformed query will be correct and more efficient than the original query. Correctness means that the transformed query must produce the same result. Efficiency means that the result of the transformed query could be computed in less time using the minimum of computer resources, e.g., disk storage and memory space.

Query optimization uses rewriting techniques for reordering the operators in the relational algebra expressions and selecting a good implementation for each of them.

In the next sections we discuss two broad classes of optimization techniques: *logical optimization* and *physical optimization*. Due to the lack of space we limit our discussion to the optimization of queries in centralized database management systems. Other query optimization issues are beyond the scope of this thesis, e.g., heterogeneous query optimization, object-oriented query optimization, temporal query optimization, and parallel query optimization [FMV93, JMP97, SJS01, MD03]. On the one hand, logical optimization performs transformations without considering physical characteristics of the underlying database. There are also two categories of such optimization: *algebraic optimization* and *semantic optimization*. While algebraic optimization is limited to the rearrangement of query operators, semantic optimization relies on semantic constraints to reformulate queries. On the other hand, physical optimization considers implementation aspects for improving the performance.

2.3 Algebraic Query Optimization

There are two kinds of transformations in algebraic query optimization: *algebraic transformation* and *rule-based transformation*.

2.3.1 Standard Algebraic Transformation

Algebraic optimization utilizes rules of algebraic operators to transform an algebraic expression into logical equivalent expressions. Reducing the amount of intermediate results involved by single operators in the operator-tree is the main motivation of this optimization. For instance, the join of smaller relations is more efficient than the join of large ones or the execution of a join is cheaper than the execution of a cartesian product. The heuristics used for this optimization are based on the rearrangement of algebraic operators and their arguments in algebra expression, and the substitutions of some operators with others that might involve less costs during the evaluation phase. To perform such transformations, a set of algebraic rules for algebraic operators are applied. The application of these rules lead to expressions which are equivalent to the original one since the rules produce equivalent transformations.

Given three relation schemas $R_1(A_1 \dots A_n)$, $R_2(B_1 \dots B_n)$, and $R_3(C_1 \dots C_n)$ where A_i, B_i, and C_i $(1 \leq i \leq n)$ are attribute names. Let p, q, p_i be predicates. $p_i(A_i)$ and $p_i(A_i, B_j)$ denote the application of p_i to attribute A_i or to attributes A_i, B_j, respectively. Table 2.1 provides a classification of the algebraic rules.

The application of these rules might lead to several significant optimizations. On one hand, some rules can be applied separately for any transformation. The most useful of them are those that push selections and projections through joins and cartesian product (rules in 8 and fourth rule in 6). In fact, $\times$ and $\bowtie$ operations are the most expensive to compute compared to other operations and reducing the size of the intermediate relations is necessary for any optimization. For instance, by applying the first rule of the category 8 only those tuples satisfying the predicate p will be combined, hence less amount of storage space will be used. In addition, this rule permits to scan the relations only once. Similarly, rules in class 6 and 7 permit to make the size of the

R_1: **Commutativity of $\bowtie$, $\cup$, $\cap$ and $\times$**	R_2: **Commutativity of σ**
$R_1 \bowtie R_2 \equiv R_2 \bowtie R_1$ $R_1 \cup R_2 \equiv R_2 \cup R_1$ $R_1 \cap R_2 \equiv R_2 \cap R_1$ $R_1 \times R_2 \equiv R_2 \times R_1$	$\sigma_p(\sigma_q R_1) \equiv \sigma_q(\sigma_p R_1)$
R_3: **Associativity of $\bowtie$ and $\times$**	R_4: **Idempotence of σ and Π**
$R_1 \bowtie (R_2 \bowtie R_3) \equiv (R_1 \bowtie R_2) \bowtie R_3$ $R_1 \cup (R_2 \cup R_3) \equiv (R_1 \cup R_2) \cup R_3$ $R_1 \cap (R_2 \cap R_3) \equiv (R_1 \cap R_2) \cap R_3$ $R_1 \times (R_2 \times R_3) \equiv (R_1 \cap R_2) \times R_3$	R: Relation over attributes S and $S' \subset S$; $S'' \subset S$ - If $S' \subset S''$ then $\Pi_{S'}(\Pi_{S''}(R)) \equiv \Pi_{S'}(R)$ - $\sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \equiv \sigma_{p_1(A_1) \wedge p_1(A_2)}(R)$
R_5: **Commuting σ with unary operator**	R_6: **Commuting σ with binary operators**
$\Pi_{A_1 \ldots A_n}(\sigma_{p(A_p)}(R)) \equiv \Pi_{A_1 \ldots A_n}(\sigma_{p(A_p)}(\Pi_{A_1 \ldots A_n, A_p}(R)))$ Note that if $A_p \in \{A_1 \ldots A_n\}$ then the last projection on $\{A_1 \ldots A_n\}$ appearing on the right side of the formula is useless.	$\sigma_{p(A_i)}(R_1 \cup R_2) \equiv \sigma_{p(A_i)}(R_1) \cup \sigma_{p(A_i)}(R_2)$ $\sigma_{p(A_i)}(R_1 \cap R_2) \equiv \sigma_{p(A_i)}(R_1) \cap \sigma_{p(A_i)}(R_2)$ $\sigma_{p(A_i)}(R_1 - R_2) \equiv \sigma_{p(A_i)}(R_1) - \sigma_{p(A_i)}(R_2)$ $\sigma_{p(A_i)}(R_1 \bowtie_{p(A_i,B_j)} R_2) \equiv \sigma_{p(A_i)}(R_1) \bowtie_{p(A_i,B_j)} R_2$ $\Pi_C(R_1 \bowtie_{p(A_i,B_j)} R_2) \equiv \Pi_A(R_1) \bowtie_{p(A_i,B_j)} \Pi_B(R_2)$
R_7: **Commuting Π with binary operator**	R_8: **Pushing σ through $\bowtie$ and $\times$**
A, B: Sets of attributes from R_1 and R_1, and $C = A \cup B$, $A_i \subset A$ and $B_j \subset B$ $\Pi_C(R_1 \times R_2) \equiv \Pi_A(R_1) \times \Pi_B(R_2)$ $\Pi_C(R_1 \cup R_2) \equiv \Pi_A(R_1) \cup \Pi_B(R_2)$ $\Pi_C(R_1 \bowtie_{p(A_i,B_j)} R_2) \equiv \Pi_A(R_1) \bowtie_{p(A_i,B_j)} \Pi_B(R_2)$	$\sigma_{p(A_i,B_j)}(R_1 \times R_2) \equiv R_1 \bowtie_{p(A_i,B_j)} R_2$ $\sigma_{p(A_i)}(R_1 \times R_2) \equiv \sigma_{p(A_i)}(R_1) \times R_2$ $\sigma_{p(A_i)}(R_1 \bowtie_{q(A_k,B_j)} R_2) \equiv \sigma_{p(A_i)}(R_1) \bowtie_{q(A_k,B_j)} R_2$

Table 2.1: Algebraic Equivalences

involved relation as small as possible before performing any operation. Furthermore, rules such as the first rule in class 4 enable to reduce the number of operands in an operation. This can imply a decrease of I/O while executing the query in further steps.

On the other hand, some rules might not be useful unless they are combined with others. For instance, the second rule in class 4 can be combined with the rule in class 8 for optimizing expressions such as $\sigma_{p_1(A_1) \wedge q(A_i,B_j)}(R_1 \times R_2)$ where predicate q involves attributes of both R_1 and R_2. This expression can be transformed as follows:

$\sigma_{p_1(A_1) \wedge q_{(A_i,B_j)}}(R_1 \times R_2) \equiv \sigma_{p_1(A_1)}(\sigma_{q(A_i,B_j)}(R_1 \times R_2)) \equiv \sigma_{p_1(A_1)}(R_1 \bowtie_{q(A_i,B_j)} R_2)$

Moreover, we note that there are rules (Classes 1, 2 and 3) which are useful in conjunction with other approaches for optimization which can capture information about physical storage of each relation involved in the operation as we shall see in Section 2.5. For instance, suppose the size of R_2 is significantly smaller than the size of R_1, then it is better to apply rules 1 to scan R_2 before R_1 while using nested loops strategies for implementing operations. However, the application of these rules to a canonical algebraic expression might lead to a huge number of equivalent expressions. Thus, it is difficult for the optimizer to determine which expression is the optimal one. A brute force method is to generate all possible expressions and to estimate the cost of each relevant plan. This approach is unrealistic because it would incur great execution time overhead for the optimizer. For instance, a query involving a join of N relations can be transformed into $N! \times f(N)$ equivalent expressions just by applying only commutative and associative rules for the join operations. Thereby the

number of relation permutations is $N!$ and the number of ways the relations can be associated is $f(N)$ (ordering of parentheses in the query sequence) [Dat02]. To make query optimization practical, the optimizer must investigate only a subset of all possible plans and must approximate their costs. Therefore, optimization approaches must rely on a set of heuristics to reduce the search space of these plans. In the following, we outline some typical heuristics [SMK97]:

1. Break up the conjunctions of selections using idempotence rule (see Class 4). This produces a set of selection operators which can be applied separately.
2. Pushing selections through joins and cartesian products down into the operator-tree as far as possible using rules of Classes 2 and 8.
3. Combine selections with cartesian products to form joins using rules of Class 8.
4. Rearrange the order of join operations to obtain an order that involves the smallest as possible intermediate relations (see Class 3).
5. Cascade projection operators and push them down into the operator-tree as far as possible using rules of Classes 4 and 7.
6. Introduce projections if possible.

Example 2.5 By applying the algebraic optimization to the query of the Example 2.3 many different trees might be found equivalent to the initial one. For instance, Figure 2.3 shows three equivalent trees. The first tree is obtained from applying Rule 4 to the initial tree to "disassociate" the selection into multiple ones. Then, Rules 2 and 6 are applied to push selections down the tree. The second tree is generated from the first one by applying Rule 8 in order to replace cartesian product operators by join operators that might be evaluated more efficiently. Finally, the reconstructing of the second tree leads to third tree by applying Rule 3 and following the Heuristic 2. Thus, the resulting tree is better than the others in the sense that the most selective operation is done first, hence the selection operations on `EMP` before the join might reduce the size of the operand relation. □

2.3.2 Rule-based Transformation

Extensible database systems like STARBURST [HFLP89], EXODUS [GD87], and VOLCANO [Gra94] have been developed to support non-standard applications (e.g., geographical applications) for which new data-types, new operations, and non-standard query languages are required and hence complex queries are formulated. Adapting traditional optimizer to such requirements is difficult and building additional functions on the top of the DBMS is inefficient [HFLP89]. Therefore, for extensible database systems the query optimizer must minimize any implementation changes needed and making optimization extensible is the most challenging aspect in its construction. The first effort towards an extensible query optimization was the use of rule-based techniques

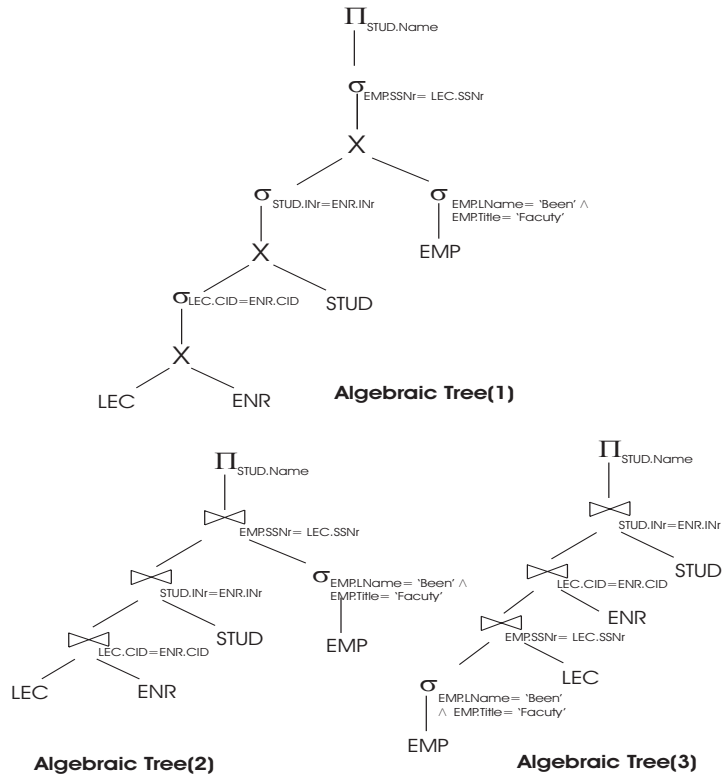


Figure 2.3: Three trees equivalent to tree in Figure 2.2

for query optimization. This approach was initially suggested by Freytag [Fre87]. Algebraic transformations used for the optimization are expressed as rules. Rule-based optimization is attractive because it allows the definition of new transformation rules which are specific to non-standard applications, and to change and to extend the existing transformations and their implementations.

As an example of an extensible optimizer we shortly describe that of the STARBURST database system. The STARBURST-optimizer consists of two components: a *rewriting engine* and a *plan optimizer*. The task of the rewriting engine is to reformulate a submitted-query to an equivalent one for better performance by using a set of rewriting rules. The task of the query planer is to choose the query evaluation plan (QEP) by determining which execution strategy is better than others for accessing the database. Therefore, dividing the optimizer architecture into two components makes it easier to perform changes and extensions of one part of the optimizer without affecting the other part. The query reformulation is done by using an abstraction model called a *query graph model* (QGM) [PLH97]. QGM is a structural representation of an SQL query used during the query reformulation process. In a QGM, nodes represent query blocks and edges represent references to the used relations across the blocks. Each query block provides information about the structure of query results as well as information about the operations required to compute these results. The query engine uses rules to transform a QGM into another equivalent QGM. A rule is expressed as a pair of condition-action. Rules are grouped together into rule classes each of which

has a particular control strategy. A control strategy specifies how rules in a class are selected for firing. There are basically three classes: predicate migration, projection push-down, and operator merging. Rules for predicate migration allow predicates to be moved down into lower level of the query tree in order to minimize the amount of data retrieved. Predicates could also be replicated and migrating replicas to multiple operations could reduce execution costs. Rules for projection push-down allow to avoid the retrieval of useless attribute values of relations. Rules for operation merging are applied to merge QGM nodes, implying merging of all relation expressions and existential subquery conjuncts consisting of restriction, projection and join, whenever it is possible. These rules allow more scope for optimization. In addition, STARBURST's rules deal with complex SQL operations such as UNION, INTERSECT, EXCEPT, and GROUP BY. The rewriting engine handles rules using forward chaining [Ull88]. It assigns a parameter to each rule, called budget, to keep track of the available resources invoked for its execution. The budget is also used to control the time spend on generating QGMs. If the budget is exhausted the query generation stops at a consistent QGM, i.e., each rule transformation is complete [PLH97].

Cherniack and Zdonik [CZ96] presented a strategy for rule-based optimizers to simplify the representation of more complex queries such as nested-queries. The strategy allows to transform nested queries into simple queries with nest of joins. In fact, rule-based system face difficulties in expressing some complex transformations (e.g., normalization). These difficulties arise due to the procedural nature (using codes) of the rules which do not permit transformations to be simple and efficient for understanding and reasoning. For instance, a transformation to convert Boolean expressions to *conjunctive normal form* (CNF) could not be expressed with a rewriting rule because patterns are too constraining to capture its generality [CZ96]. In [CZ98], Cherniack and Zdonik suggested an approach that enables the rules to be declarative by avoiding the use of codes. Their approach is based on the combinator algebra (a variable free-algebra) for representing queries in a flexible form so that additional codes to rewrite rules are no longer required for performing complicated transformations (e.g., subquery rearrangement). This is done by removing variables from a query expression. Furthermore, the proposed representation may simplify reasoning about the meaning of query transformations.

2.4 Semantic Query Optimization

So far, we presented query optimization techniques that are based on syntactic transformations to generate logical equivalent queries. In this Subsection, we describe another technique which is based on semantic transformations to generate semantic equivalent queries. Two queries are said to be semantic (or logical) equivalent if their answers return the same results in any database state. This technique is called *semantic query optimization* (SQO). SQO was first introduced by King [Kin81] and by Hammer and Zdonik [HZ80]. The key idea of SQO is to modify the logical structure of a query (e.g., adding or deleting restrictions) without modifying the query answer but it could be processed in more efficient way. Since a SQO-based optimizer uses knowledge about

the content of a database the search space of equivalent queries in SQO is much lager than in its counterparts in conventional syntactic optimization. For this reason, it can detect more optimization opportunities and provide higher cost reduction.

It is important to note that logical equivalent queries are obviously semantical equivalent but semantical equivalent queries do not need to be logically equivalent. That is, two semantical equivalent queries might return different answers if the current state of the database violates some semantic rules. For example, suppose we have a constraint on the relation `Student` that enforces student ages to be greater than seventeen years old. Thus, a query that asks for students whose age is less than seventeen years old will return an empty result set. However, if the the constraint is violated and data are inserted into the relation for students whose ages are eighteen years old, then the answer to the query will not be the same [Kin81].

The research on query optimization covers several aspects of SQO. These aspects include:

- **The type of semantic knowledge exploited for the optimization.** SQO approaches use different types of semantic knowledge for transforming queries. Semantic knowledge [1] capture the meaning of data in the underlying database. They can be categorized into two groups: *integrity constraints* (ICs) and *ontological knowledge* (OK). While integrity constraints are explicitly defined in the database, ontological knowledge can be derived from an external source, e.g., ontology (see Section 3.3).

- **The type of database system under consideration.** SQO are applied on different types of database systems. Several approaches have been developed for relational databases [Kin81, SO89, SY94, CGK+99] and deductive databases [YS89, CGM90, SY94, Hsu96]. In the context of object databases, most of the work done on SQO adapt or extend early techniques from deductive databases [YK93, HR95, GG+97, BBS03].

- **The type of query under consideration.** One of the important issues on deductive databases is optimizing recursive queries [LH89, LH91b, LM95]. Chakravarthy et al. proposed a technique, called *semantic compilation* , for reformulating queries in a preprocessing step prior to any query evaluation [CGM90]. The technique is based on the use of fragments of ICs, called *residues*, which are relevant for the optimization. Residues are extracted through subsumption of each database clause by an integrity constraint or part of it [CGM90]. Moreover, in order to transform non-recursive queries that contain join/unions of predicates, a number of techniques based on ICS have been developed [LHQ91].
  There are two well-known techniques for optimizing recursive queries. The first technique uses methods proposed by Chakravarthy et al. [CGM90]. For example, it applies the residue method to each subquery inside the evaluation loop in a bottom-up manner [LH89].

[1] See chapter 2 for the definition of this term

The second technique converts sub-expressions of a query into Datalog programs. Some of these sub-expressions could be removed using integrity constraints. Removing redundant expressions results in reducing the time needed for evaluating a query because the number of joins could also be reduced during the evaluation. One of the algorithms that detects and removes redundant atoms of query expressions is described in In [Sag88].
The mentioned approaches broadly address problems related to conjunctive queries or queries that can be translated to a union of conjunctive queries [CV92]. Extending SQO to other cases, in which queries are more complex such as queries involving aggregation and duplicates, remains a challenging problem [LS95].

- **The type of algorithms for generating semantically equivalent queries.** The main goal of SQO is to correctly generate semantically equivalent queries for a query with respect of given ICs. Several algorithms have been developed to derive the appropriate ICs for the transformation process. These algorithms can be classified into *logic-based* algorithms [JCV84, CGM90], *graph-based* algorithms [LH91a, LM92], and *hybrid-based* algorithms. Logic-based algorithms use a logic formalism to represent queries and ICs as logic rules and deduction is used to produce equivalent query forms. Chakravarthy et al. [CGM90] assume that database components and queries are expressed in Datalog and perform subsumption for the transformation process. Graph-based algorithms use a graphical formalism to represent and reformulate queries. Lakshmanan and Hemandez [LH91a] developed an optimization algorithm based on hypergraphs for transforming recursive queries. The algorithm is used to detect and eliminate redundant subgoal expressions in the query trees based on the functional dependencies. Jarke et al. [JCV84] use Prolog as framework and apply tableau simplification algorithms for the optimization. Lakshmanan and Missaoui [LM95] proposed a method as a hybrid of logic-based and graph-based approaches for a class of linear recursive queries. The method generates a set of ICs using a modified form of the subsumption algorithm [CGM90] and push them inside the recursion of a query to obtain more efficient queries.

- **Control strategies for limiting 'promising' candidate queries and filtering useless information.** The choice of which transformation to use for SQO leaving out a transformation may miss an important candidate query, performing irrelevant transformations may increase the number of equivalent queries and hence increases the optimization costs. Several approaches have addressed this problem. However, not all the semantic information embedded in the database are useful for the optimization. Some approaches use transformation heuristics [Kin81, SO87, CGM90] to limit the number of candidates while others use learning techniques [YS89] to specify most appropriate semantic knowledge rules necessary for the optimization in a given database state. The latter approaches can be classified into query-driven [YS89, SSS92] or data-driven approaches [SHKC93, Hsu96, SL97]. In query-driven framework query transformations for a new query arriving at the database are guided by the set of queries previously executed. In the data-driven framework query transformations are di-

rected by the data leading to the learning of rules which characterize patterns in the data that represent the transformation.

2.4.1 Categories of Integrity Constraints

Most of the work in SQO deals with a subset of ICs [Xu83, JCV84, SO89]. We classify ICs based on the states of a database as shown in Figure 2.4. ICs fall into two categories: *static integrity constraints* (SICs) and *dynamic integrity constraints* (DICs). SICs refer to the constraints that must hold for all database states whereas DICs refer to the constraints that must hold for a specific database state [YS89]. That is, DICs do not need to be always true but are frequently satisfied by the database state. While SICs are defined by the users and/or database administrators DICs could be captured from the database during query executions. Yu and Sun [YS89] proposed an method for deriving dynamic constraints during query processing by comparing and analyzing the results of queries previously processed. The constraints are applied to subsequent queries for optimization. Siegel et al. [SSS92] presented a learning-method to automatically derive algebraic rules for the optimization. The rules are drawn up from the intermediate results during the optimization process. They give the optimizer the capability to learn such rules from the the database usage pattern (frequencies of query, update, insert, delete, and complexities in the qualification portions of query and data manipulation statements).

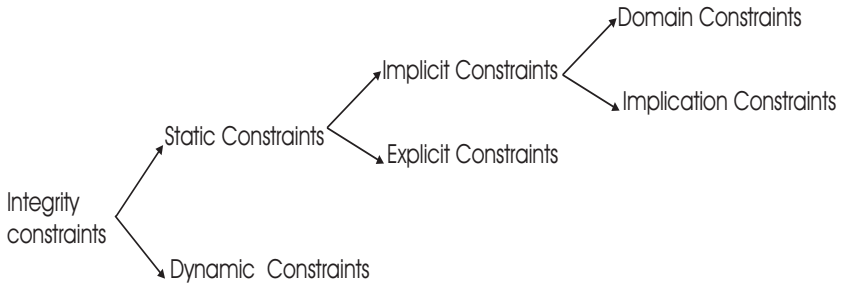


Figure 2.4: Categories of Integrity Constraints

Static ICs can be classified as *implicit integrity constraints* and *explicit (semantic) integrity constraints*. Explicit ICs are application-oriented constraints (e.g., business rules). They specify some requirements of the application in concern. These constraints could not be represented in the database schema. They are explicitly specified in the database using a specific language. For instance, an organization may specify that the department head's salary should be higher than that of any other employee of the department. Implicit ICs are implicitly represented in the database schema, e.g., the data type of values allowed for an attribute. The mentioned SQO techniques use different types of implicit ICs. Basically, the integrity constraints are characterized by two features of dependencies: *inclusion dependencies* and *functional dependencies*.

1. **Inclusion dependencies (IDs):** They define a superset-subset relationship between values of two different set of attributes of two relation schemas (not

necessary distinct). Formally, let X and Y be two sets of attributes of two relation schemas R and S, respectively. The inclusion dependency is defined as $\pi_X(r(R)) \subseteq \pi_Y(r(S))$, it denotes that attribute values of X in R exist in the relation S as values of Y . Inclusion dependencies is an important aspect of the inter-relational constraints (e.g. referential integrity constraints) [JCV84].

2. **Functional dependencies (FDs):** They define other types of relationships between two sets of attributes from a database [Kin81]. Formally, let X and Y be two sets of (distinct) attributes of a relation schema R. We call the dependency $X \rightarrow Y$ a functional dependency of Y on X. That is, for any two tuples t_1, t_2 $\in R$, if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$, i.e., the values of the Y component of a tuple in $r(R)$ depend on (or determined by) the values of the X component. This definition is extended to specify a non-equivalence operator θ in the implication $t_1[X]\,\theta\,t_2[X]$ and to consider dependencies for more attributes from more than one relation schema.

In the following we describe some kinds of integrity constraints discussed in the literature. The difference between these types of constraints is small. However, we find it appropriate to describe their features and clarify the terminologies used by the authors. We present in Table 2.2 a cross-reference among these constraints, their features and their supported techniques in the literature.

- **Domain constraints:** This type of constraints involves only one attribute. Domain constraints restrict the domains of attributes either with a constant or with another attribute. The restriction may use any comparison operator [Xu83]. For example, they may restrict integer values of an attribute to a given subrange. King [Kin81] refers to this kind of constraints as *domain definitions*.

- **Value constraints:** They are a subset of domain constraints. They specify the lower and upper bound values of particular domain.

- **Bounding constraints:** This type of constraints involves more than one attribute. Bounding constraints imply a condition on a given attribute if the conjunction of the conditions on other attributes is satisfied.

- **Dependency constraints:** These constraints are a subset of bounding constraints. They assume that all attributes must be included in the same relation schema.

- **Production constraints:** They provide inter-relational constraints between two relations. Production rules have the form of a conjunction of conditions on various attributes that implies a condition on another attribute [Xu83]. The conjunction could be coupled with a join condition involving an attribute of each relation. King [Kin81] refers to this kind of constraints as *restrictions on relationships*.

- **Referential integrity constraints:** They are a subset of the production constraints. They ensure existential dependencies between values of two attributes

of different relation schemas. That is, the values of a given attribute in a relation must also appear as values of a key-attribute in another relation.

- **Implication constraints:** They define a restriction on the relative domains of two or more attributes of the same or different relation schemas. They are formally defined as a clausal rule (a horn clause with no positive literal and one or more negative literals) composed of conjunction of negated predicates. These predicates could be relational predicates or evaluable predicates [2]. These constraints are referred to as *production rules* by King [Kin81].

- **Subset constraints:** They ensure the inclusion dependency between two attributes of two different relation schemas [SO89]. The definitions of these constraints are only restricted to domains and do not include existential qualification in the implication as compared to referential integrity constraints [CFP82].

	[Xu83]	[JCV84]	[SO89]	[Kin81]	[CGM90]	Features
Domain constraints	✓			✓	✓	FDs
Value constraints	✓	✓		✓	✓	FDs
Bounding constraints				✓	✓	FDs
Dependency constraints	✓		✓	✓	✓	FDs
Production constraints	✓		✓	✓	✓	FDs
Referential ICs	✓	✓		✓	✓	IDs
Implication constraints			✓	✓	✓	FDs
Subset constraints			✓		✓	IDs

Table 2.2: Cross-reference of the Integrity Constraints supported in the literature

2.4.2 Transformation Heuristics

Like syntactic optimization techniques, SQO techniques use various heuristics for query transformation in order to reduce the number of alternative queries which are produced by applying the rules. We describe the heuristics that are deemed good in the literature for SQO as follows.

1. Restriction elimination (T_1): A restriction (qualification) is redundant if it is satisfied by the consequent atom of an implication ICs. Removing redundant restrictions may reduce the number of join operations in a query.

2. Restriction introduction (T_2): The idea is to reduce the number of inner scans of join operations, assuming the join is implemented as nested-loop. By introducing an additional literal we might restrict one or both of the relations involved by the join. This transformation is highly effective when the join involves large relations. This transformation is referred to as *scan reduction* by King [Kin81].

[2] Evaluable predicates represent comparison between (i) a variable and a constant, or (ii) two variables, or (iii) two variables and an offset (see [Ull88]).

3. Literal enhancement (T_3): The idea is to substitute an evaluable predicate in a query by more restrictive clauses that might be inferred from ICs and hence we may reduce the size of the relation to scan.

4. Literal elimination (T_4): The idea behind this transformation is that if we eliminate a literal clause from a query by using ICs then we may be able to eliminate a join operation as well. This transformation is only useful when the relation involved in the join does not contribute any attribute in the query's answer. This transformation is referred to as *join elimination* by King [Kin81], Shenoy and Ozsoyoglu [SO89], and Xu [Xu83].

5. Join introduction (T_5): The idea based on this heuristic is that it might be advantageous to introduce a join with an additional relation if that relation is relatively small compared to the initial relations of the query. This transformation is also called *literal introduction* in [CGM90].

6. Index introduction (T_6): This transformation introduces a constraint on an attribute of a relation used in the query which is indexed [Kin81]. Indexes make query execution faster.

7. Detection of unsatisfiable conditions (contradictions) (T_7): A contradiction in a qualification implies a null answer to a query and therefore access to database is not necessary.

2.4.3 Classification and Comparison

Several classifications of SQO approaches could be done by considering each aspect of the optimization separately. We propose a classification based on the semantic knowledge used for SQO as shown in Figure 2.5. Maintaining our focus on centralized databases we classify SQO approaches into three classes: *static constraints-based*, *dynamic constraints-based*, and *ontological-based* approaches. For each class, the figure lists the main features of different techniques that are used.

The approaches in the first class use integrity constraint which can provide knowledge about the structure of a database and how its content is organized. Such knowledge could be extracted from the database (database catalog). For instance, knowledge about attributes such as indexes are useful for the optimization. Constraints about indexed attributes could be used to substitute other constraints in a query and hence reduce execution costs. In [Kin81] King illustrated how to use knowledge about file structures in form of clustered indexes to optimize queries. Generally, the proposed approaches for SQO using static integrity constraints describe query optimization as two-phase process. In the first phase, a query is transformed into semantically equivalent queries. In the second phase the conventional query optimizer applies further optimization to the queries resulting from the first phase [CGM90].

The approaches using dynamic integrity constraints are based on the argument that static integrity constraints may not be used for optimization in some cases. In addition, dynamic constraints which are frequently satisfied by the database state allow to "hunt"

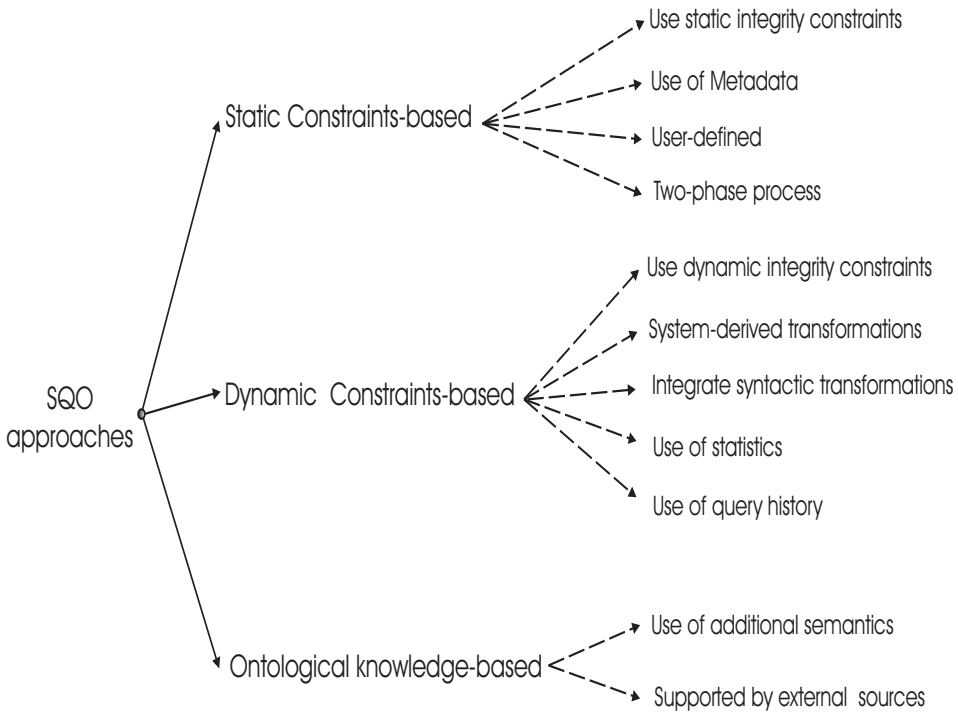


Figure 2.5: Classification of SQO approaches

for more opportunities for optimization. That is, adding dynamic constraints as input to SQO might could create new search spaces in order to find effective queries. However, the set of dynamic constraints generated by the system may become large and some constraints may be useless for generating equivalent queries. To overcome this problem dynamic ICs based approaches use learning techniques to limit the number of applicable constraints. A constraint is considered applicable to a query if the antecedent part of its expression is a logical consequence of the query literals. Such techniques use information about the query history and performance statistics. The history of executed queries allows to identify characteristics of constraints suitable for the optimization. The past performance of the optimizer allows to determine the value of existing constraints. Statistics obtained from the past constraints can be used to determine factors in the estimation of candidate constraints for a given query. Example of such factors are derivability, maintenance and antecedent selectivity [SSS92]. For instance, Sayli et al. developed a statistics-based method to estimate which attributes in a constraint are correlated [SL97].

Another important aspect of derived ICs-based approaches is the integration of SQO and conventional query optimization in one phase. One of the advantages of this integration is that it allows interactions between algebraic and semantic transformations in order to reduce the search space. For instance, algebraic heuristics (e.g., hill climbing) can be used to cut off unnecessary transformations before they are produced [SSS92].

Table 2.3 summarizes the main feature of different techniques for SQO that have been covered in this section. To date to our knowledge there are no approaches which use ontological knowledge about data for semantic query rewriting in centralized data-

	Type of OK	Type of ICs	Data Model	Query Language	Query Type	Logical Rewriting	Query Representation
[Kin81]		SICs	Relational	SODA [a]	SPJ	T_2,T_4,T_5, T_6	Graph-based
[Xu83]		SICs	Relational	SQL	SPJ	T_4,T_6	Graph-based
[JCV84]		SICs	Relational	Prolog	SPJ [b]	T_1,T_3,T_7	Graph-based
[SO89]		DICs	Deductive	Datalog	SPJ	T_1,T_2,T_4, T_6	Graph-based
[CGM90]		SICs	Deductive	Datalog	SPJ	T_1,T_2,T_3, T_4 T_5,T_6, T_7	Logic-based
[LM92]		SICs	Deductive	Datalog	Recursive	T_1,T_4	Graph-based
[SSS92]		DICs	Relational	QUEL	SPJ	T_2,T_4, T_7,	Graph-based
[LM95]		SICs	Deductive	Datalog	Recursive	T_1,T_2, T_3, T_7	Hybrid-based
[HK96]		DICs	Deductive	Prolog	SPJ	T_1,T_2,T_3, T_4, T_7	Logic-based
[GG+97]		SICs	Object	Datalog	SPJ	T_1,T_3, T_4, T_5, T_7	Logic-based

Table 2.3: Summary of SQO techniques's features for centralized DBMSs

[a] It is a form of relational calculus

[b] It is a function free conjunctive query

base systems. We argue that (static) integrity constraints alone are not sufficient for SQO. Integrity constraints capture a small part of the database semantics. This is because they are primarily defined to check the consistency of data not for other purposes. However, data in most of current databases, for example real-life databases, are rich of semantics, but few of them are expressed as integrity constraints. Moreover, dynamic integrity constraints do not represent "robust" knowledge. That is, they are not always useful for optimization because they concern only a snapshot of database instance. After a database is changed, some derived constraints may become inconsistent with a new database state. Therefore, this lack of sufficient semantics may be one of the reasons why SQO has not become prominent for commercial database systems. We believe that additional semantics in form of ontological knowledge could provide effective enhancement of early SQO techniques. This issue may open a new research area of query optimization.

In this thesis we present an approach similar to SQO. We attempt to use ontological knowledge to reformulate queries not for the purpose of optimization but rather to obtain meaningful answers as described throughout this thesis. The previous approaches state the condition that the transformed queries must be equivalent to the original ones. We argue that this condition should be relaxed in many situations where a user does not need only "exact" answers for their queries but those answers that better meet his intention. We define an exact answer as being the set of database items which literally match the terms of query predicates. In fact, responses to queries may not provide information the user really wants. A user may need additional information or even different information than the query requests. Furthermore, he may prefer an alternative answer to his queries over not receiving any answer at all [LNR97].

One of the interesting feature of our approach is that query reformulation are represented as rewriting rules. These rewriting rules are devised in a manner similar to that used for rule-based optimizer. The approach could be easily implemented in extensible

database systems because of the high extensibility nature of their query processor as illustrated earlier. We essentially deal with SPJ-queries and develop methods used for transforming query predicates.

2.5 Physical Query Optimization

Physical query optimization [JK84, FMV93, Ioa96, Cha98] is complementary to logical optimization. The application of logical query optimization results in a set of semantically logical equivalent queries. For each query several access plans can be generated. These plans specify how each algebraic operator is physically implemented. In fact, an algebraic operator will be mapped into one or more *physical operators*. For instance, the join operator can be implemented as nested loop or merge scan or hash join. The execution cost of a physical operator depends on properties of data in the database such as available indexes and size of the involved relations. To physically optimize a query the following steps are applied:

- Determine the adequate physical operators and generate candidate evaluation plans.
- Determine the overall cost of each plan.
- Choose the possibly "cheapest" plan.

Like in logical optimization, one of the difficult problem for physical optimization approaches is how to reduce the search space of plan candidates since the number of generated plans can be too large. To cope with this problem heuristic methods are needed. Execution plans are examined and compared by the optimizer based on the cost models of physical operations and the size and distribution estimation.

2.5.1 Transformation Heuristics

These methods determine the choices of implementing algebraic operations. The choices are related to data structures that support implementation and the evaluation of join orders. For instance, it is typical for an optimizer to choose a particular tree shape, called *left-deep trees*, when a query expression involves a sequence of join operations. A tree shape corresponds to the parenthesizing algebraic expression that involves an associative and commutative operator. A plan which corresponds to a left-deep tree is called a *left-deep plan*. Thus, the query shapes a, b, c, and d represented in Figure 2.6 correspond to the expressions $A \bowtie B \bowtie C \bowtie D$, $(A \bowtie B) \bowtie (C \bowtie D)$, $A \bowtie ((B \bowtie C) \bowtie D)$, and $((A \bowtie B) \bowtie C) \bowtie D$, respectively. The later tree shape corresponds to a left-deep tree expression (Figure 2.6(d)). Left-deep trees have their right child of any join being always a base relation. A tree is called *bushy tree* if it contains at least one join between two intermediate results. Query optimizers usually choose an access plan containing left deep-trees over other plans. Left-deep tree are desirable because of two reasons: First, such a tree restricts the inner operand of a join to a

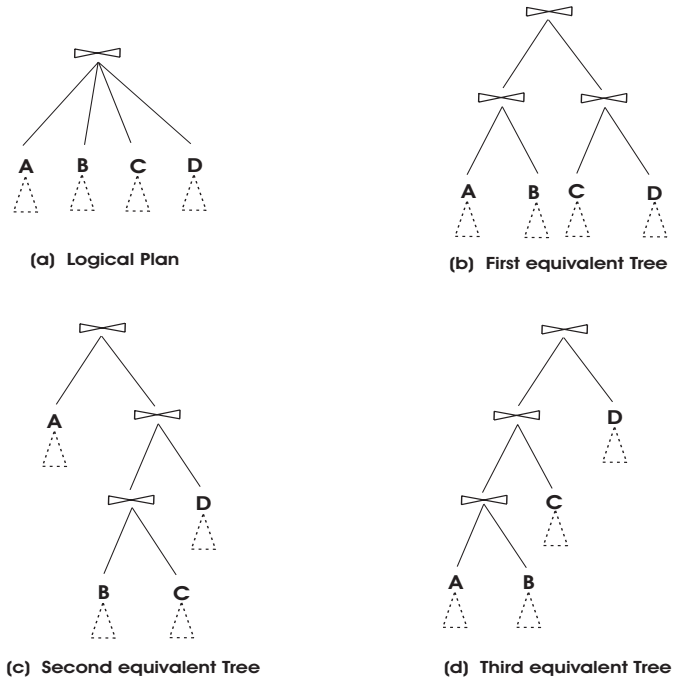


Figure 2.6: Logical Plan and three equivalent Trees

base relation and hence reduces the need of materializing intermediate results. Thus, joins can be executed in a *pipeline* fashions, for example, in Figure 2.6(d) we can first compute $A \bowtie B$ and pipe the result to the next join with C and the result of the next join can further be piped up to the tree without storing intermediate relations in the disk. Second, there are nonetheless plenty of left-deep plans to choose from and so the best among them will likely suffice. Third, the space of plans containing bushy trees is vastly larger and hence much more expensive to search [VM96]. As a consequence of choosing only left-deep trees the number of candidate plans containing join trees will be significantly reduced. For instance, for SPJ-queries containing N relations the number of left-deep plans is in the order of $O(2^N)$ in comparison to $\Omega(N!)$ of all alternative plans [Ioa96].

Moreover, there are several algorithms used for searching the best plans. They are classified into: *randomized algorithms*, *genetic algorithms*, *hybrid algorithms*, and *deterministic algorithms*. Deterministic algorithms apply deterministic steps or exhaustive search strategies. One kind of such algorithms is called *dynamic programming algorithm* which is used in a number of commercial DBMs (e.g., DB2, Ingres, SYSTEM R). The best known algorithm has been developed in the context of SYSTEM R [SAC$^+$79] by Selinger et al. The algorithm takes as input a logical plan corresponding to a set of N-way join-expression $E_1 \bowtie \ldots \bowtie E_N$, where E_i is a 1-relation expression, and provides a "cheap" left-deep plan as follows. The algorithm first evaluates the cost of all plans for computing each argument of the join-expression, called *1-relation plans*. We note that each 1-relation expression might have multiple plans since there might be different access paths (e.g., various indexes). The best plans (plans with lowest costs) among all these 1-relation plans are expanded into 2-relation plans, then 3-relation plans, etc.

In order to expand a best 1-relation plan (say plan p_{i_1} for E_{i_1}) into a set of 2-relation plans, p_{i_1} is joined with every 1-relation plan. Then, the cost of all such plans are evaluated and the best 2-relation plans are retained. Similarly, each best 2-relation plan (say $q_{i_{12}}$ for $E_{i_1} \bowtie E_{i_2}$) is expanded into a set of 3-relation plans by joining $q_{i_{12}}$ with every 1-relation plan. Again, only the best plans (plans with lowest costs) are retained for the next stage.

Randomized algorithms perform *random moves* in the search space using pre-defined set of probabilistic rules and terminate as soon as no more applicable moves exist or a time limit is exceeded. A search space is viewed as a graph in which each node contains a query tree. The moves constitute edges between the different nodes of the graph such that two nodes are connected by an edge if they can be transformed into one another by exactly one move.

Generic algorithms resemble to randomized algorithms in their probabilistic foundations. Their basic idea of search is inspired from the biological evolution: A random set of populations, each with its own cost, and pairs of populations are matched to generate offspring by random crossover and mutation. Those populations with the least cost survive the selection for the next generation. The algorithm terminates when there is no further selections. The last survived population is considered to be optimal. Hybrid algorithms mix the strategies of deterministic, generic, and randomized algorithms: Solutions obtained by deterministic algorithms are used as starting points for randomized algorithms or as initial population members for genetic algorithms [SMK97].

2.5.2 Cost Models

In order to estimate the cost of a plan we need to estimate the cost of resources for computing every operator in the plan such as

- Working storage requirements,
- I/O costs, and
- CPU costs.

Cost estimation techniques rely on *cost models*. Cost models are arithmetic formulas that are used to approximate the cost of a plan taking into account the available computation resources. Most cost models primarily focus on the secondary storage access and various distributions of data in relations. Usually, cost models need statistical information about properties of data and indexes (e.g., number of data pages in a relation, number of pages in an index, number of distinct values, cardinality) and existing access methods (e.g., sequential vs. random access). This information is maintained in the system catalog of a DBMS. We should notice that the tradeoff in such estimation technique is the generation of more efficient plan (based on estimated costs) versus the optimization overhead. Thus, a desirable optimizer is one that efficiently controls the searching costs and trades optimization time with the quality of the execution plan [SSD92].

2.6 Summary

In this chapter we presented the theoretical foundations that are most relevant to our approach. In particular, we gave an overview of query processing in database systems and presented a survey of the rewriting techniques proposed in the literature. We focused our attention on the various forms of query transformation applied during the query processing phases (i.e., syntactical, logical, and semantical forms). In particular, we discussed semantic query rewriting and proposed a classification of the techniques used for this approach. As we have seen these techniques use semantic knowledge in the form of integrity constraints for the rewriting process. Ontological knowledge about a database content has not been considered. Moreover, most of the existing approaches have mainly concentrated on enhancing system performance (e.g., execution time). Nevertheless, database management systems also need techniques to improve the quality of the query answer since computing resources (e.g., CPU time) are no longer bottleneck costs for many of the current systems. Furthermore, users are more interested in answers that better fit their needs. In Chapter 5, we propose a new technique that is targeted to responde these requirements.

Chapter 3

Semantic Knowledge

Investigating, representing and exploiting *semantic knowledge* of data are critical issues for many applications in computer science. In particular, in database systems and information systems research issues related to semantic knowledge have been extensively studied to solve problems that are caused by increasing amount of data in information sources, heterogeneity of their representations, disagreements on the interpretation of their meaning, and degradation of their quality. Although several approaches have been developed for this purpose, researchers have different perspectives on the nature of semantic knowledge and their use. In this chapter, we illustrate the meaning of terms such as "*semantics*", "*syntax*", and "*models*" in order to distinguish between representation issues and semantics issues. We introduce *ontologies* as an important source of semantic information which will be the focus of discussion in this thesis. We address the problem of using ontologies for data management and argue that explicit representation of semantics is needed for database query processing. In particular, we introduce the problem of answering queries in a single relational database system using additional semantic information.

In the following sections, we first illustrate basic concepts related to semantics. Many researchers make no distinction between semantics and syntax and there are others who combine them under the term "*models*". A clear distinction is needed to define semantic formalisms such as ontologies and to discuss semantic problems the thesis is going to address. Section 3.1 illustrates the difference between terms such as syntax, semantics, and models. Section 3.2 introduces semantic models and emphasizes their role in organizing data in information sources. It intends to provide a background understanding while sections 3.3, 3.4, 3.6 and 3.5 describe ontologies in general and compare them to other conceptual formalisms. Section 3.3 defines ontologies, describes their characteristics, and Section 3.4 illustrates their potential benefits and advantages in data management. Section 3.5 clarifies the difference between ontologies and other conceptual formalisms such as conceptual schemas. Ontologies and conceptual formalisms have closely related concepts and hence are confusedly used in the literature. The distinction between them will not only clarify what ontologies are but also illustrate why we need them to solve semantic problems. The solution to the problem presented in this thesis is based on these distinctions. We deal with query processing using ontologies in single database environments.

3.1 Syntax, Semantics, Models

Meaning of Things. In the logical and philosophical fields *semantics* is the study or science of meaning. That is, the study of relationships between *signs* and what they represent [Dic00]. A sign designates a "thing" that refers to a real-world concept. For example, a sign can be a picture which is composed of a knife and a fork. This picture is internationally used to indicate a restaurant. The American philosopher Peirce [Pei58] identifies three types of signs: *icons*, *indices*, and *symbols*. Icons provide forms similar to something, like circles to denote a sun. Indices are traces of something, like fingerprints to denote a finger. Symbols are what we usually term sings. They describe something according to some conventions. Words and alphabetics are symbols. We should note that most of the symbols have no natural connection with things they describe (i.e., no meanings in themselves). For instance, there is nothing in the sound or visual of the word "star". On one hand, a symbol might have different meanings. The literal meaning of "star" is a planet visible for us in the universe at night. However, a "star" is also someone who is dazzlingly skilled in a particular field.

On the other hand, the same thing [1] might have different symbols. For instance, the words "morning star" and "evening star" are two distinct symbols but refer to the same thing, the planet Venus. The first symbol means a star seen in the morning while the second symbol means a star seen in the evening. Therefore, symbols are associated with a third participant, our though, to make sense of the things they refer to. In Figure 3.1 we present the three basic relationships evoked in the meaning triangle as cited by Ogden and Richards [OR23].

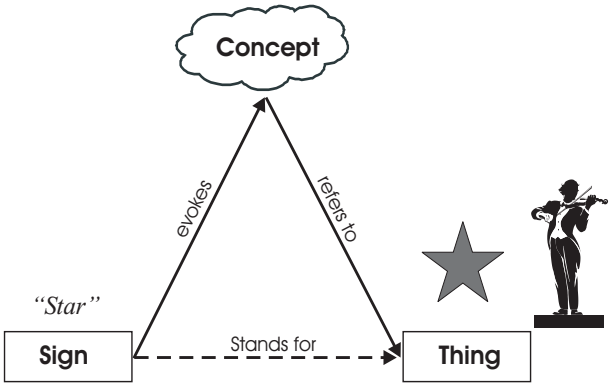


Figure 3.1: The meaning Triangle

On the right of Figure 3.1 is the thing represented by an icon that resembles a star. On the left is a symbol, the word "star", that stands for this thing. There is no direct relationship between sings and things. When we employ a sign, we perform an act of reference to a particular thing or experience. When we speak (or listen) "s-t-a-r" we might think about a planet. This thought is called *concept* (displayed on the top of the triangle). Therefore, a concept is the unique intermediate that relates a symbol to a thing.

[1] Things mean whatever is physical or abstract.

It is important to note that when some people use signs to communicate their intentions to others about things in the world, they should agree on common sings to refer to the same things. Thus, the relationships between signs and what they represent (things) depends on the persons who interpret those signs and in which situation or for which purpose they are used for. For this reason, Peirce defines sings more specifically by including these additional aspects: *a sign is something which stands to somebody for something in some respect* [Pei58]. For instance, the sign "morning star" might create distinct interpretations in human minds. For people who live on the Earth "morning star" would refer to the planet Venus whereas for people who are on the planet Mars it would refer to the planet Jupiter. We say that a sign is interpreted in a *context*.

Signs by themselves are not sufficient for the communication between agents (e.g., humans, machines). That is, a language is needed. Languages can be natural languages (using words, phrases, etc.) or constructed languages (using pictures, geometrical forms, sounds, etc.). In general, a language consists of a set of signs and syntactic rules. The rules define the relationships that signs have among themselves and the ways in which they can be manipulated. They are used to build statements (sentences) about things in the world in such a way that these statements are valid. Therefore, a language can be viewed as a set of all statements that conform to its rules. In this context, the rules constitutes the *syntax* of a language. In Morris's classic sense, semantics concerns two domains: a domain of signs which are governed by syntax rules and a domain of things the signs "mean" [Mor38]. These domains are called *syntactic domain* and *semantic domain*, respectively. Accordingly, semantics can be defined as the correspondence between these two domains [Rap58]. It is what links concepts with sings in the meaning triangle (see Figure 3.1).

Meaning of Data. In database and information systems, researchers are concerned with *semantics of data*. In [Woo75], Wood generally defines data semantics as *the meaning and use of data*. Accordingly, data semantics concerns the interpretation of data stored in the system. Data are symbols that represent some part of the real world, called usually *the mini-world*. That is, a subset of the world that includes only the entities of interest for an application. In fact, the meaning of data is captured by the model of some agents that agree on a common representation of the mini-world. Such an agreement could be either formal or informal. In addition, semantics concerns the usage of data. That is, the meaning of data depends on the applications that use these data. Applications represent only the aspects of real-world entities that are relevant to them. For this reason, Rosenthal considers regularities in databases (i.e., constraints) which capture regularities in the real-world as components of data semantics [She95]. That is, applications determine in which context data are used. Hence, Sowa provides a more precise definition of semantics [Sow84]. He states that

"*the common aspect that unifies all the groups [of specialists] is a knowledge of the meaning of the data and the constraints necessary to keep it a faithful model of the real world. The study of the meaning and constraints on the data is called database semantics*".

Semantic issues are extensively addressed by approaches developed for managing data in multiple information sources. Critical management tasks are to define a uniform

view over multiple schemas and to formulate queries using multiple languages. Usually, these schemas have heterogenous formats (structured, semi-structured, or unstructured) and use different models (relational model, object-oriented model, etc.). The key success in performing data integration is to identify objects [2] in information sources that have the same meaning or are semantically similar. For instance, an attribute in one schema, e.g., EMPLOYEE might denote the same set of entities as an attribute in another schema, e.g., WORKERS. While some approaches rely on features of objects such as names, key attributes, and the context in use, to capture semantic similarity, other approaches apply probability measures to evaluate similarity [SJB96, Lin98]. The former approaches attempt to determine semantic aspects of relationships between pairs of objects with respect to their meaning in the real world. For instance, Sheth et al. [SK93] provide three types of relationship: *semantic equivalence*, *semantic relationship*, *semantic relevance*, and *semantic relevance*. There, semantics are captured in terms of relationships between objects using the context to which the corresponding objects belong. However, the latter approaches resort to additional knowledge source (e.g., lexical thesaurus, dictionaries) for quantifying similarity weights betweens source concepts that corresponds to object names. Hence, these weights are used for understanding the semantic relationships between the objects [GR04].

In summary, the major issues related to semantics are:

- Synonymy: Different ways two express the same meaning.
- Homonymy: Different meanings of the same symbol.
- Ambiguity: To have more than one meaning of a symbol in the same context, e.g., for some people the word "Adult" refers to humans who are more than 18 years old whereas for others it refers to humans who are less than 18 years old. We note that homonymy is a special case of the ambiguity.
- Context: Indicates the relationships "surrounding" the meaning of a symbol.
- Ontological relationships: Complex relationships between real world objects might exist. Semantics of such relationships are often described by specific knowledge sources, called *ontologies*, as we shall see in Section 3.3.

Models. When we attempt to describe a domain (or a situation) we conceive a *model*. A model is an abstraction of the real world related to a situation. The philosopher Smith [Smi87] calls every act of conceptualization, analysis and categorization a "process of abstraction". For instance, a genetic blue-printing would be concerned with the structure and the relationships of its beams but not be with the arrangement of proteins in the wood of which the beams are made [Smi87]. To design a model, we use a collection of symbols and build a structure that we consider to comply to our understanding of the world. This structure is called a *model description (representation).*

[2] The term object refers to schema elements of a source, e.g., attributes in relational model. In contrast, the term entity refers to subject matter in the real world

It is important to note that it is not possible to completely model a world situation. All that we can do is to provide a model of a fragment of the world that we think is important in the respective context. The world around us is so complex that much people trying to understand it may get lost in its infinite richness [Rap58]. According to Smith, models are inherently *partial*. For this reason, it is not surprising that agents usually provide different model representations for the same domain of real world. They represent only a fragmentary point of view of the reality of a complex domain. Thus, there is a gap between the full reality of the world, on one hand, and the model and their representation, on the other hand, insofar that the latter fails to capture the full semantics of the former. To bridge this gap Smith suggest to understand a model in terms of other models. This constitutes what Smith calls a "correspondence continuum". This continuum can be considered as a chain of domains that correspond to one another where each of them (except the last) is understandable in terms of the next. The last domain is the actual real world entities. In fact, "correspondence" is one of fundamental principles of understanding: To understand something is to understand it in terms of something else. That is, if we face a new phenomenon or experience we attempt to understand it by looking for something known for us with which we compare the new phenomenon or experience. A good example that illustrates the "*correspondence continuum*" was cited by Smith [BM04]: The modeling of a ship S. Consider a photo P_1 of S and assume it is a model of S. In addition, consider a photo P_2 of P_1. Although P_2 is not a photo of S the information about S is still preserved. Thus, P_2 can be used as a model of (the model of) S. Similarly, if P_2 is digitized then the digital photo of P_2 can be considered as the model of (the model of the model of) S. We consider this paradigm to define ontologies and to compare them with conceptual schemas in Sections 3.3 and 3.5.

3.2 Semantic Models

Over the last decade, a great variety of formalisms for representing models of applications have been developed in artificial intelligence, software engineering and databases. These formalisms, called *semantic models*, differ in their degrees of expressiveness as well as in their notations.

Semantic models offer a set of expressive facilities to capture abstractions about the application domain in terms of *semantic units* such as *entity*, *activity*, *goal*, and *agent*. Moreover, they offer means for structuring abstractions in terms of *abstraction mechanisms* such *classification*, *generalization* and *aggregation*. Due to limited space, we briefly present some models and focus on these aspects. An extensive overview of this topic can be found in [Myl90, RP00, BB03].

Models in Artificial Intelligence. Artificial intelligence applications turned out to require representations of a great deal of human knowledge in order to act "intelligently". As a result, they require formalisms for representing the real world in a way close to the human mind. The first formalism was conceived by Ros Quillian in his PhD thesis [Qui68]. This formalism is called *semantic network* and has been originally

designed for modelling the structure of human memory. A semantic network is a form of directed labelled graph with different kinds of nodes and edges. Figure 3.2 depicts a simple semantic network representing some animals (taken from [BCN+03]). Semantic networks offer two semantic primitives for the modelling process: *concepts* and *associations*. That is, nodes in the graph represent *concepts* of words (word senses) and edges represent *associations* between them. For instance, the sentence "shark eats human" is represented in a semantic network by two nodes labelled by the words "shark" and "human" and linked by an edge of type " `Eat`". For words with specific meaning nodes are linked with edges of type " `ISA`". Moreover, semantic networks allow for edges of type " `AND`" and " `OR`" between nodes and each concept could have associated properties such as "can fly" for the "bird" concept. The main feature of semantic networks is that a property is inherited along " `ISA`" edges unless it is modified in more specific concept. For instance, canards inherit the property "can fly" from birds whereas ostriches do not inherit this property because it is modified for them. A detailed description of semantic networks can be found in [Sow91].

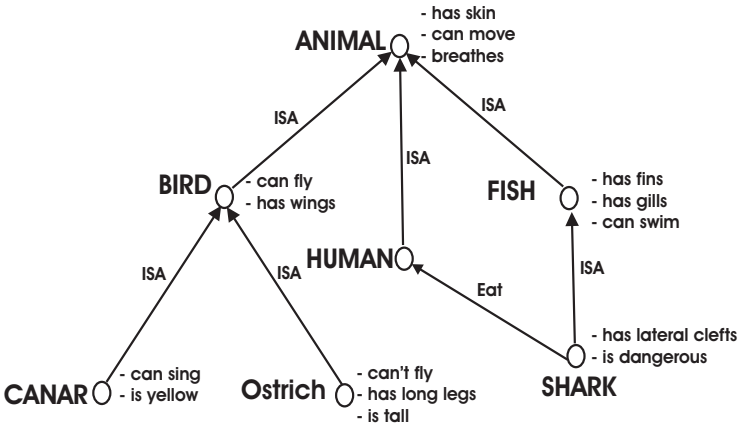


Figure 3.2: A semantic Network for Animals

Following Quillian's memory model several variants and extensions of semantic networks have been proposed. Among them are *conceptual graphs* and *frame systems*. Conceptual graphs (CGs), which have been conceived by Sowa [Sow84], are a kind of semantic network. CGs are popular for representing semantics, especially in natural language processing. A CG is a bipartite graph which have two kinds of nodes called *concept nodes* and *conceptual relation* nodes. Each edge in the graph links a concept node and a conceptual relation node. Figure 3.3 depicts a conceptual graph of the sentence "*Marry is going to France by plane*" [Sow00]. This graph consists of four concepts: `Go`, `Person:Mary`, `Country:France`, and `Plane` and three conceptual relations: `Agnt` relates `Go` to the agent Mary, `Dest` relates `Go` to the destination France, and `Inst` relates `Go` to the instrument plane. CGs offer new semantic primitives in building the graph including the typing of concept labels. Indeed, some concepts are generic, i.e., they have only a type label. Other concepts are individual, i.e., they have a colon after the type label following by a name or a unique identifier. Besides these features, CGs allow translating semantics into a logical form (first order

formulas). The representation of logical formulas as CGs has been proven to be adequate for their manipulations. In fact, CGs allow reasoning about these formulas. For example, deciding whether a given graph is valid, i.e., whether the corresponding formula is valid or whether a given graph g is subsumed by another graph g', i.e., whether the formula corresponding to g implies the formula corresponding to g'.

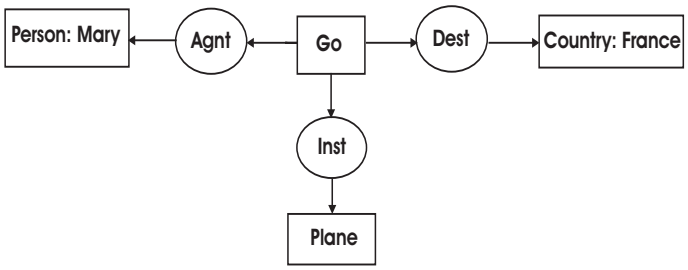


Figure 3.3: A conceptual Graph

On the other hand, *frame systems* have been proposed later by Minsky [Min74] based on the semantic network paradigm. They rely on the notion of *frame* as a symbol structure and on the capability of expressing relationships between frames. A set of related frames are defined and represented by frame systems. Frames are considered more suitable for representing concepts than CGs [SCM03]. They contain information about the components of a concept being described, relationships to similar concepts as well as procedural information on how the frame can be accessed and changed over time [Min75]. For instance, given the following frame definition:

Frame BioCourse in **KB** University
SuperClasses: Course
MemberSlot: taughtby
ValueClass: Professor
Cardinality.Min: 1
Cardinality.Max: 1

In this definition, `BioCourse` denotes the name of the frame. It represents biology courses which are taught by at least one professor. Biology courses are also instances of the frame `Course`. The frame system is called `University`. `Taughtby` is a slot name which denotes a frame relation to another frame called `Professor`. We should note that, in general, the value of a slot can be an expression of a constraint on multiple frames. Moreover, a frame might have multiple slots. The cardinality restrictions that are imposed to the number of professors who teach the courses are denoted by `Cardinality.Min` and `Cardinality.Max`. Like semantic networks frame systems allow properties for concepts. However, properties in standard semantic networks are restricted to primitive, atomic ones whereas in frame systems they can be complex concepts. The properties are represented as *slots* in a frame. Slots are similar to entries in a record. Like CGs frame systems enable reasoning about frames. This could be achieved in two ways: (1) Using "partial matching", i.e., more specific frames are embedded into more general ones. (2) Searching for slot *fillers* (slot values) to collect more

information concerning a specific situation [BCN+03]. In the literature, we find many models and languages which have been developed and implemented using the frame formalism. Early examples of frame system are KL-ONE [SCM03], KRL [BW77], and PSN [LM77].

Models in Software Engineering. Within software engineering a large number of semantic models have been proposed for specifying requirements of software systems. The development of these models was launched by Ross [RS77] who proposed the *Structured Analysis and Design Technique* (SADT) as a method for describing activities in the real world. The SADT models the world in terms of *activities* and *data*. It uses a diagram representation in which the nodes represent activities and the edges represent flows of data between activities. There are four kinds of data: data that are consumed by activities (input data), produced by activities (output data), data that control activities, and data that represent mechanisms by which the activity is to be performed. An activity is represented by a rectangle. Arrows entering a rectangle from the left represent input data, arrows leaving to the right represent output data, arrows entering from above represent control information, and arrows entering from below represent additional mechanisms.

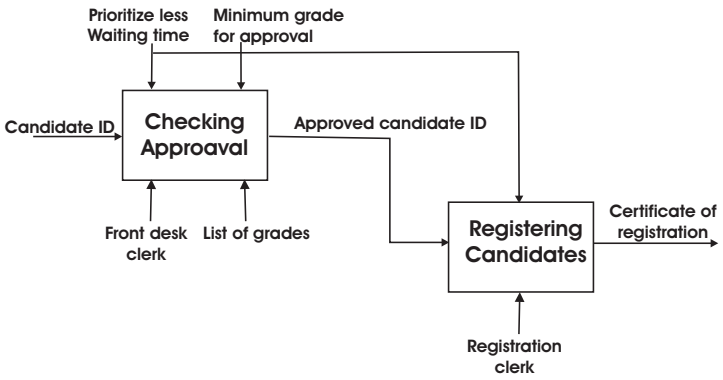


Figure 3.4: An SADT Diagram

Figure 3.4 shows a SADT diagram describing a registration process of students, taken from [LB02]. The diagram has two activities represented by rectangles. Semantically, a student will arrive at the school's secretary and will present its candidate ID to the front-desk clerk. He will check for the student's name in the list of grades and if the grade is equal or greater to the minimum grade for approval, the student may proceed to the registration clerk who will effectively enroll the student in the course, handling him out the registration certificate. It is important to note that activities can be structured hierarchically so that each activity can be decomposed into another sub-activities [Wie98].

Following Ross's proposal many other models have been developed, e.g., DFD [NWP90], IDEF [BY85], CIM [GMB94], and RML [Bub80]. A recent approach is the Unified Modelling Language (UML) [Gro05] which integrates features of its antecedents models in object oriented analysis and design. UML offers a different set of diagrams called *use*

case diagrams which represent the static and dynamic aspects of the system. A use case is a sequence of actions that an actor (usually a person or a system component) performs within a system to achieve a particular goal. The new feature of UML is that nodes in diagrams represent *object classes* where objects have a local state represented by a set of attributes and local operations. In addition, relationships such as generalization and aggregation are represented by special arrows. Other kinds of relationships between classes are represented by *associations*. In UML individual objects can be represented by their own classes. However, in UML only binary relationships can be represented. Higher arity of relationships should be represented by creating new classes. Moreover, cardinalities of attributes and other specific constraints are expressed in additional language called OCL [3].

Models in Database Systems. Within database systems semantic models, called also *semantic data models*, have been primarily introduced as a formalism for database schema design. Semantic models carry the design process by attempting to logically structure and organize data in a database in a way that can capture more meaning than the conventional logical database models. These models (hereafter called conceptual models) offer semantic primitives that are more expressive and hence are closer to the way users think about data.

There are a large number of semantic models which have been proposed over the last three decades. The Entity-Relationship (ER) model is the most widespread semantic model which has been extensively used for the design of commercial applications. Initially, the ER model was introduced by Peter Chen [Che80] and extensions have been proposed with minor differences in notation and expressiveness. The basic elements of the ER model are *entities* and *relationships*. The ER model is used to represent the conceptual structure of data in a database systems by means of an *ER schema*. The ER schema is drawn using a standard graphical ER notation in which entities are represented as rectangles and relationships as diamonds. An entity type (simply entity) denotes a set of objects (instances) of a domain of interest which have common properties. Elementary properties are modelled through *attributes*. However, these attributes do not have local operations unlike attributes in UML. Relationships between instances of different entities are modelled in relationship types (simply relationship). The relationships can have any type and any arity. Cardinality constraints can be represented to restrict the number of times and instance of a entity may participate in a relationship. However, other integrity constraints on data can not be represented directly in the diagram. Figure 3.5 shows a simple ER schema that describes an ordering of products, taken from [Myl98]. There are three entity types (`Customer`, `Order` and `Products`), and two relationship types (`places` and `contains`). Informally, the schema represent the fact that "customers place orders" and "orders contain products". `places` is one-to-many relationship ("a customer may place many orders but an order must be placed by only one customer") while `contains` is many-to-many relationship ("an order may contain many products and a product may be contained in many orders").

[3] Object Constraint Language

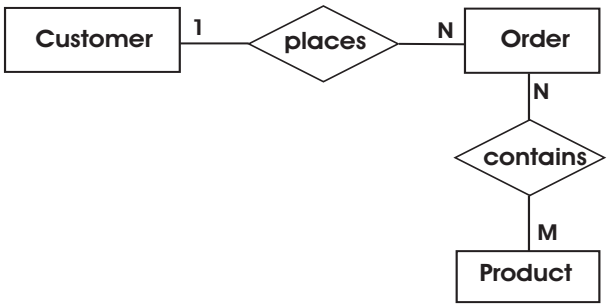


Figure 3.5: An Entity Relationship Schema

There are many other proposals for extending ER model which introduce new features such as complex types of attribute and specific kind of relationships. Examples of extended ER models are HERM [Tha93], SDM [HM96] and USM [Ram95].

It is also important to mention the emergence object-oriented data models (OODMs) which constitute the design framework of the object oriented database applications [Wie98]. OODMs have been proposed based on the object-oriented programming approaches. The central feature of OODMs is the object. Objects can represent things in the real world as in the ER model but they can also represent operations or processes on other objects. A set of objects that have the same structure (attributes) and behavior (operations) are represented by a class called *object class*. The object schema is a conceptual representation of the object model in the same way an ER schema represents an ER model. OODMs have similar primitives like ER models and the correspondence between the ER model and the corresponding object data model can be easily determined. The major difference is that objects must include the encapsulation of data operations as well as the data structure which is expressed in the object class definition. The ER model, on the other hand, specifies only the data structure and ignores the dynamic behavior. Example of object data models are the O2-Model [BDK92] and the ONTOS-model [And92].

We should note that semantic models make different assumptions about the nature of the part of real world they intend to model. For instance, the ER model assumes that the real world consists of *entities* and *relationships* whereas the SADT model assumes that it consists of *activities* and *data*. Therefore, a model might be appropriate for some applications but not for others. That is, it might not be appropriate for applications which violate its assumptions. For example, temporal applications, which describe instantaneous events, need models that represent time (using sequence of time intervals or time points). The ER model is not appropriate for such applications. In general, some models are suitable for representing *static* aspects of the world others are suitable for representing *dynamic* aspects of the world.

Regarding abstraction mechanisms there are some semantic models that provide more support for certain mechanisms than others. For instance, classification is well supported in UML but not in SADT. The issue of evaluating semantic data models has been considered in [Myl98, MSBS03], among others.

3.3 Ontologies

In this section, we present ontologies as an important source of semantic knowledge that can be useful for a wide range of applications in database and information systems, software engineering, or natural language processing. We attempt to focus our attention on the database area. First, we define what an ontology is, illustrate its characteristics, and explain its role for data management. This will already reveal the great potential this approach has on our task. Afterwards, we sketch our idea of how ontologies could be used in order to support query processing. The idea presented will be elaborated in the remainder of the thesis.

3.3.1 Definition of Ontologies

In recent years, the term *"ontology"* has been intensively used in the fields of artificial intelligence, and database and information systems. However, there are many different definitions of what an ontology is. Often, we find definitions in the literature that are very general or tailored to the specific needs of an application.

Originally, the term "ontology" has been taken from philosophy where it means the study of the existence. In the artificial intelligence field, an initial definition was given by Tom Gruber: " *An ontology is an explicit specification of a conceptualization*" [Gru93]. Studer and his colleagues [SBF98] argue that a conceptualization refers to an abstraction of some phenomenon in the world. This is achieved by identifying the relevant *concepts* and *constraints* on their use that must be explicitly defined. We refer the reader to these papers [Gru93, GG95, NH97, CJB99] for more definitions.

Informally, we define an ontology as an intentional description of what is known about the essence of entities in a particular domain of interest using abstractions, also called *concepts* and *relationships* among them. This description should be *clear*, *concise*, and *consistent* for a particular community. A community may be a group of people or computer systems that interact with one another within a common domain of interest. The members of a community need support to communicate and understand each other. In the medical domain, for example, we have a set of applications that provide medical information from different sources. Members of the medical community may not be able to exchange information among these sources unless they use shared vocabularies which are specific to the domain. Data exchange needs a unique interpretation of data. An ontology might be a good support for communicating between sources in the same domain if a community must agree upon the terms used for the ontology's definition. We say that a community *commits* itself to an ontology. Such an ontology is referred to as a *domain ontology*, i.e., it is related to a specific domain (e.g., the medical domain). Domain ontologies provide vocabularies about concepts within a domain and the relationships between these concepts, about the activities that take place in that domain, and about the rules governing that domain [PFLC03]. Moreover, a community might commit to another ontology of other communities. These communities then build their own ontologies based on more general ontologies, called *Top-level ontologies*. Top-level ontologies provide a set of vocabulary terms that represent common sense and from which terms of other ontologies are specialized. Examples of such ontologies are

Ontolingua [FFR96] and Cyc [LG90].

With regard to the semantics issues discussed earlier (see Section 3.1), ontologies can be viewed as models that capture semantics about entities of a mini-world concerning a certain community (or a set of communities). The notion of community is important in the definition of an ontology since *agreements* about what the ontology should describe is necessary for the development, the representation and the use of its content. That is, every member of the community must agree on the semantics of terms in the common ontology used to describe properties of entities and how they are related, hence, to decide which aspects of the world to attend to and which to ignore. Thus, these ontological commitments offer a unified view on what a community believes to be relevant because the complexity of the natural world is overwhelming. It is important to mention that *ontological commitment* is a basic criterion for building and using ontologies. The commitments represented in an ontology must be *minimal*. That is, an ontology must contain as few constraints as possible about real-world entities being represented allowing parties committed to the ontology to instantiate their applications as required. For instance, an ontology representing employees may focus on properties such as age, position and salary without specifying which values should be assigned to each employee. The salaries may range between 1800 $ and 3000 $ for one application whereas they may range between 1000 $ and 2000 $ for another. We can say that ontologies must achieve a form of *semantics independence* for the applications they use, like the form of data independence in databases. Indeed, semantics in ontologies could be used for a wide range of applications not only for one. Ontological constraints (representing a higher level of abstractive restrictions) are related directly to concept descriptions and not to particular application requirements.

It is worthy to note that most of the existing ontologies are organized in a way that they support computational inference. They contain axioms which enable reasoning about the meaning of concepts, e.g., subsumption.

3.3.2 Describing real world objects

Real world objects are described in ontologies using *concepts*, *relationships*, and *axioms* as follows.

- **Concepts** are abstractions that denote real-world objects having common properties. They are usually referred by their names, which are words taken from some natural language. These words must be unambiguous. Furthermore, concepts are organized in a taxonomy (`ISA`-hierarchy), which relates more specialized concepts (sub-concepts) to more generalized concepts (super-concepts) that subsume them. Sub-concepts related to the same super-concept are *disjunctive*, i.e., a set of entities represented by one sub-concept does not overlap with a set of entities of another sub-concept. Properties, which are captured as concepts and/or axioms in an ontology, may be of two types: *intrinsic* properties (e.g., flavor for perfume) and *extrinsic* properties (e.g., perfume's name and company's name where it is manufactured). We emphasize that in an ontology concepts are represented, not words. In general, concepts are not specific to a given natural language [BP02].

- **Relationships** They define kinds of interactions between entities of the same concept and entities of other concepts. There are two classes of relationships: *generic* relationships and *domain-specific* relationships. The first class includes specialization (`ISA`), synonymy (`SynOf`), and decomposition (`PartOf`) relationships. The second class contains relationships that are specific to the domain of ontology. Semantics of relationships are defined by axioms. Basically, the hierarchical organization of concepts through the `ISA`-relationship constitutes the backbone of an ontology. In general, the IS-A hierarchy of concepts could be represented as a directed acyclic graph such that no concept has more than one super-concept.

- **Axioms** are truth statements about concepts and their relationships. They specify properties of the real-world entities represented in the ontology and properties of relations they relate (see Section 3.6). Axioms are usually formalized using logical languages and must be consistent.

Properties of Relationships. We assume that each of the following relationships: `"ISA"`, `"SynOf"` and `"PartOf"` has the common semantics known in most information modelling systems:

- `" ISA"`: Represents a classification of the concepts. For instance, the statement c_1 `ISA` c_2 means that any instance of the class concept c_1 is also an instance of the class concept c_2. The real world objects referred by c_2 inherit all the characteristics of the objects that are referred by c_1. The `ISA`-relationship serves as the basis for property inheritance.

- `" SynOf"`: The statement c_1 `SynOf` c_2 means that the objects referred by concept c_1 could be also referred by concept c_2.

- `" PartOf"`: This relationship represents the part-whole association between a composite concept and a concept representing one of its components. For instance, the statement c_1 `PartOf` c_2 means that objects that are referred by concept c_1 are part of objects referred by concept c_2.

Properties of relationships are explicitly represented using axioms from the ontology. We use the FOL-language [4] to formulate axioms because of its great expressiveness. Axioms represents mainly the following kinds of properties:

1. *Algebraic properties* such as reflexivity, symmetry and transitivity [SM00]. We consider the `ISA`, `SynOf`, and `PartOf` relationships as transitive. For example, the transitivity of `ISA` can be expressed by the following axiom:
   $$\forall\, x\, y\, z\ Isa(x, y) \wedge Isa(y, z) \Longrightarrow Isa(x, z)$$

2. *Composition* of relationships: Relationships might be expressed in terms of each other. This property is mainly related to domain specific relationships. For

[4] First Order Language

instance, a relationship `GrandMotherOf` is composed of a relationship `MotherOf` and `ParentOf`. This composition can be expressed by the following axioms:
$\forall\, x\, y\, \exists\, z\; GrandMotherOf(x, y) \Longrightarrow ParentOf(z, y) \wedge MotherOf(x, z)$
Appendix A.2 describes the axioms that express properties of generic and domain specific relationships.

3. *Exclusivity* and *Incompatibility* between two relationships. The exclusivity between two relationships R_1 and R_2 is formalized by $\forall x\ y\ \neg R_1(x, y) \Longrightarrow R_2(x, y)$ and the incompatibility is formalized by $\forall x\ y\ \neg R_1(x, y) \wedge R_2(x, y)$. For instance, `ParentOf` and `MotherOf` are exclusive, and `MotherOf` and `FatherOf` are incompatible.

Definition 3.1 (Ontology) *We define an ontology as a triple* $O = \{\zeta, \Re, \Im\}$*, where*

- ζ *is a set of concepts. They have names defined by a finite set of non-null strings* $\zeta = \{c_1, \ldots, c_n\}$*,*
- $\Re$ *is a set of relationships. They have different types defined by a finite set of non-null strings* $\Re = \{\beta_1, \ldots, \beta_n\} \cup \{$ `"ISA"`, `"SynOf"`, `"PartOf"`$\}$*, where* $\beta_i \in \Re$ *represents the product of two subsets* ζ_n *and* ζ_m *of* ζ*,* $\beta_i : \zeta_n \times \zeta_m$*. Furthermore,* $\zeta \cap \Re = \emptyset$*, and*
- $\Im$ *a set of axioms for defining properties of relationships.*

Assumptions. It is important to note that many researchers already combine concepts with instances of real world entities for specifying their ontologies [GG95]. In our definition we do not consider instances as components of an ontology. In addition, an ontology does not hold cardinality constraints about how many entities of one concept can be related with an entity of another concept. We argue that these constraints must be part of the application specification but not a part of an ontology [NF03].

3.4 The Role of Ontologies

The rapidly increasing amount of data in database and information systems has made semantic knowledge essential to support their applications. Ontologies have proved to be a good candidate to assist in the tasks of data management. In this section, we want to illustrate the role of ontologies and reveal their great potential with respect to our work.

Content Explanation. Ontologies by definition are primarily used to explain aspects of the real world. They explicate the meaning of terms used for describing some domain. In addition, they can be used for explicating the content of information sources if they are related to the same domain. While information sources model extensional information about entities of a part of the world using a collection of terms, ontologies offer a means to understand these terms. In this context, ontologies can act as a

meta-data layer over underlying sources by specifying the terminology (and its meaning) used for the stored data. That is, they can map terms occurring in the data source (syntactic domain) to their corresponding entities in the real world (semantic domain). For instance, information sources usually contain specialized terms used for particular applications such as e-commerce and geographical applications. These terms often occur in a hierarchical form: instead of describing all characteristics of an entity represented in the source a single term is used to relate entity to a class of entities that share the same characteristics. Ontologies are a good candidate to support this kind of applications since classifications are their main concern. In e-commerce, for example, products are classified into groups of categories. Ontologies for products can provide descriptions of such categories and offer standard vocabularies to refer them. UNSPSC [Com03b] and ECLASS [Com03a] are examples of such ontologies.

Content Enrichment. Ontologies may provide additional semantics about the source content. Since meaning of vocabularies of an underlying source is restricted to the services it offers, hence only limited aspects of represented entities are described. For instance, a source representing devices of a manufacturer might provide structural information about devices such as their components. However, an ontology describing concepts of devices may also provide topological aspects such as the physical behavior of devices, i.e., how the components interact inside a system. PhysSys [Bor97] is an existing ontology that can be used for this purpose. Acquiring "more semantics" about content of information sources is useful to retrieve information from them. In this thesis, we show the benefit of ontologies for querying relational databases. We use additional semantics to improve the quality of answers to user queries.

Moreover, semantics of information sources are not explicitly represented in sources. They are embedded in various levels. For instance, semantics in databases are embedded in the data model, data structures, constraints on data, and data domains. When a record contains a term "water", there is more than one way to look at this term. We can think about its potential use (a fluid necessary for the life) or its chemical aspects (a compound consisting of hydrogen and oxygen). Ontologies have been implemented to overcome the implicit hidden semantics by making the conceptualization of the information source explicit. Mapping source elements (structure and data) to ontology components can assist a computer system to understand the meaning of the source content and to define the context of its terminology. Furthermore, representing semantics in ontologies at a high level of formality makes them *machine-processable* (the KIF [Gen91] language was initially defined for this purpose). The machine-processable semantics sets the basic foundations for the next generation of the World Wide Web, the *Semantic Web* [BLHL01]. In our work, we derive additional semantics from ontologies and use them for specifying user queries.

3.5 Ontologies versus Conceptual Formalisms

In the literature, ontologies are sometimes confused with other conceptual formalisms including *Controlled Vocabularies*, *Taxonomies*, *Knowledge Bases*, and *Conceptual Schemas*.

This is understandable, since they have many conceptual features in common, as explained below.

Controlled Vocabularies (CVs) are commonly used in the field of linguistics, to describe a set of standardized terms with commonly agreed semantics for a specific domain within users communicate [LHGP99]. A special kind of controlled vocabulary is a *thesaurus*. Thesauri provide terminological knowledge of a given domain, and as much "knowledge" as ontologies. Thesauri describe a large set of special terms used in a certain domain with an explanation of their meanings. They organize these terms into groups and subgroups, and relate them with each other through relationships of linguistic nature. For example, the *Broader-Than* (`BT`) and *Narrower-Than* (`NT`) relationships indicate that a term has broader meaning than another term and vise versa. Although these relationships have the same meaning as the specialization (`ISA`) relationship and its inverse, respectively, they have sometimes the same meaning as the decomposition (`PartOf`) relationship [PM01]. Moreover, thesauri may use a relationship of type *Related-To* (`RT`) to indicate that two terms are related to each other. However, this relationship has ambiguous meaning since all relationships relate terms. In general, an ontology contains more relationships which are clearly defined and formally specified compared to a thesaurus. Ontologies deal with concepts which are defined in a way independent of a particular natural language [LG90]. However, thesauri are dependent on a specific natural language (or multiple language in case of a multilanguage thesaurus).

Taxonomies refer to classifications of things, whether they are physical or abstract, in a tree structure according to the subclass/superclass paradigm. Thus, there is only one type of relationship relating concepts representing these things, namely the `ISA`-relationship. We can conclude that a taxonomy is a kind of ontology but an ontology is usually much more than a taxonomy.

For the AI-community *A knowledge base* (KB) consists of two parts: an intentional part which describes intensional knowledge about the domain and an extensional part which describes extensional knowledge about specific entities of a domain. Intensional knowledge is a form of concept definitions expressed in logical language. Extensional knowledge exists in the a form of assertions which map instances of entities onto their corresponding concepts and specify their relationships. One of the main characteristics of KB systems is that they can infer implicit knowledge about real world objects using reasoning tools. Thus, the intensional part of a KB plays a role similar to that of an ontology. However, intensional knowledge usually depends on the application (or problem) for which the KB has been developed whereas an ontology does not depend on a particular application.

A conceptual schema is the result of modelling a mini-world based on a conceptual model as seen in Section 3.2. Similar to ontologies, conceptual schemas capture data semantics of the mini-world at a certain abstractive level which includes only semantics that are relevant to particular requirements of an application. These requirements have a great influence on how the resulting schemas look like and what they contain. In comparison, ontologies capture the meaning of the mini-world entities in a way independent of any application requirement. This application-independency is the main difference between ontologies and conceptual schemas. Nevertheless, we can say that

ontologies complement semantics that are represented by conceptual schemas. Application designers may rely on ontologies to derive their conceptual schemas. This can be done by selecting some relevant concepts from an ontology and adding constraints that are necessary for their instances. Concepts and relationships of a given conceptual schema might be initial components for creating an ontology [Mee01]. In short, if we consider semantics as a correspondence continuum as described in Section 3.1, conceptual schemas can be viewed as domains of the continuum and an ontology is the single anchor that is related directly to the intended real-world. That is, each schema has a correspondence(s) to one or more other models in the continuum and the ontology is the last model in the continuum as illustrated graphically in Figure 3.6. Elements of schemas could be directly or indirectly mapped to those of the ontology.

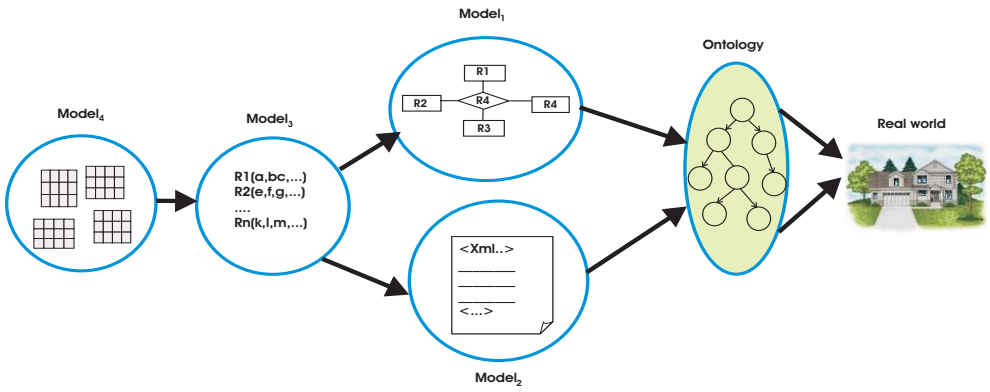


Figure 3.6: Semantics Continuum

Conceptual schemas focus on the structural aspect of data representation since they are mainly used for organizing data in a manner oriented towards certain data models. For example, ER-schemas are used to implement relational databases. However, ontologies focus on the semantical aspect of the represented entities, i.e., understanding the concrete meaning of concepts, in order to guarantee a consistent interpretation for the committed communities.

From a practical point of view, conceptual schemas are preserved in off-time mode. That is, once a conceptual schema is designed and the corresponding database is implemented, it will be no longer used. In contrast, ontologies could be formally shared and exchanged at run-time [JDM02]. Moreover, ontologies provide *reusable* semantics. Shared vocabularies from ontologies could be used during the exchange of information between different applications. As mentioned by Uschold [UG96] reusability and reliability are two benefits for system development that can be derived from using ontologies. A conceptual schema for an application does not support reusability because it is oriented to a particular application domain. Furthermore, a conceptual schema uses vocabularies that are not intended to be shared with other applications.

3.6 Ontology Representation

3.6.1 Kinds of Representation

Ontologies can be expressed in different modelling techniques including those described in Section 3.2. The choice of an appropriate semantic model to represent an ontology depends on the purpose for which the ontology is build and the underlying assumptions for achieving these goals. Ontologies, which attempt to represent a hierarchical classification of concepts, may only need models that provide typed relationships with rather intuitive semantics. Examples are ontologies that present concept-taxonomies using `ISA`-relationships. However, ontologies, which enforce more constraints, may need more formal models that can provide more expressive primitives. Examples are ontologies that need models having reasoning mechanisms to infer knowledge from ontology contents. Furthermore, model assumptions made by a particular semantic model must conform to those assumptions made by the ontology. As mentioned in Section 3.2, every semantic model offers particular build-in semantic primitives that restrict our view of the world to its own modelling perspective. That is, every kind of representation provides its own terms of what it is important to attend and assumes that anything not formulated in these terms may be ignored. Therefore, selecting a semantic model that could be appropriate for an ontology can be viewed as a form of ontological commitment for the ontology designers [UG96]. For instance, if an ontology intends to describe only static aspects of the world and assumes that it consists of a set of entities then the ER-model could be appropriate to represent such an ontology. Hence, the resulting ontology would perceive the world being modelled as entities and relationships neglecting other aspects of the world such as dynamic aspects. In addition, every semantic model constrains the interpretation of an ontology content and what knowledge could be implicitly inferred. For instance, a semantic network considered to represent a family ontology, which describes family memberships, should not contain a cycle in its links.

It is important to mention that selecting a particular model for representing an ontology influences the selection of the language that will be used to formally implement that model. Basically, these languages are based on frame approaches (e.g., FLogic [KLW95]) or first-order logic approaches (e.g., KIF[Gen91], Ontolingua [FFR96], LOOM [Mac91]) or even both (e.g., CycL [LG90]). In addition, a new class of languages has been developed for Web applications using ontologies. These languages, called *ontology markup languages*, include SHOE [HHL03], OIL [FHvH$^+$00], DAML+OIL [Hor02], and OWL [MH06]. All these languages vary in terms of their degrees of expressiveness and their capabilities of supporting reasoning.

3.6.2 Graph-based Representation

In this section, we present a graph-based formalism to represent ontologies. We introduce its basic formal settings and some related operations that are necessary for identifying concepts and relationships in an ontology.

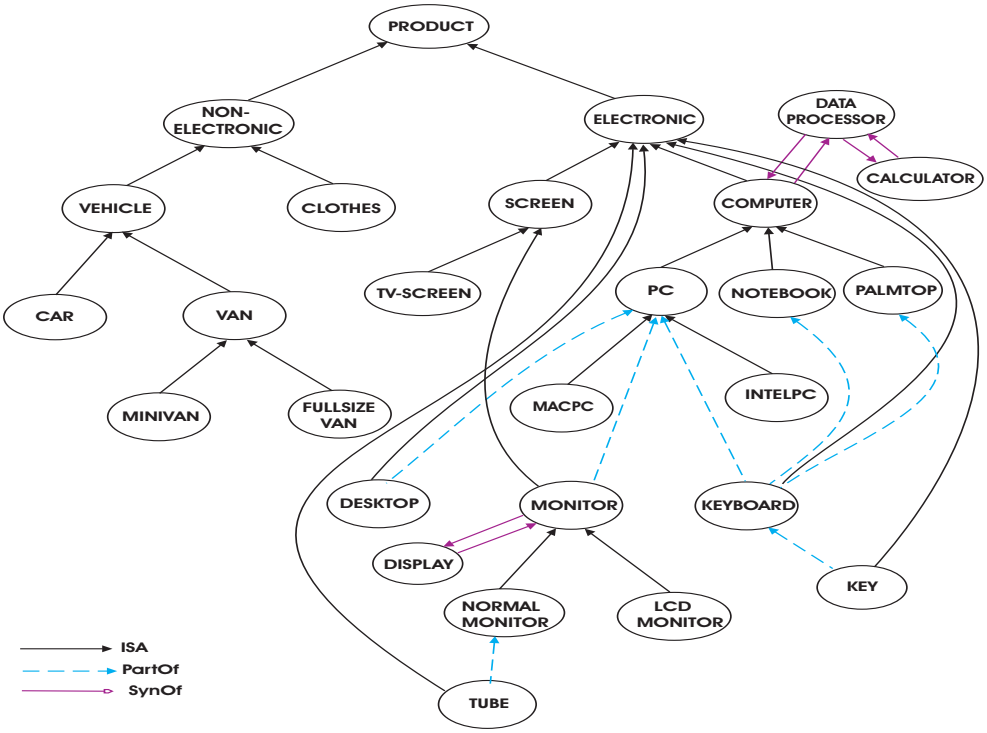


Figure 3.7: Product Ontology (PO)

Informal Description. An ontology $O = \{\zeta, \Re, \Im\}$ can be represented as a directed labelled graph $G(V, E)$, where V is a finite set of vertices and E is a finite set of edges: Each vertex of V is labelled by a concept from a set ζ and each edge of E is labelled by an inter-concept relationship from a set $\Re$. Further, we refer to a node by its label (a concept) and refer to an edge by its two node concepts and its label (a relationship). We refer to $e = c_1\, R_i\, c_2$ as the edge between two concept nodes c_1 and c_2 which is labelled by the relationship R_i of $\Re$. Mathematically, the ontology graph G can be represented by a relation $G \subseteq \zeta \times \zeta$, which is equal to the union of each relation R_i. Hence, an expression of the form $(c_1, c_2) \in R_i$ refers to $c_1\, R_i\, c_2$.

Figure 3.7 gives an example of a graph-based representation of a fragment of an ontology called "Product Ontology" (denoted by PO). The ontology describes concepts and their relationships related to products. A part of this ontology is adopted from an ontology described in [KC01]. In the following we extend the definition of an ontology as introduced in Section 3.3.

Definition 3.2 (Ontology Structure) *An ontology structure is a 4-tuple* $O = \{G, \zeta, \Re, \Im\}$, *where*

- ζ *is a set of concepts,*
- $\Re$ *is a set of relationships (distinct from* ζ*),*
- $\Im$ *a set of axioms for defining properties of relationships, and*
- G *a graph for representing inter-relationships between concepts.*

Graph Operations. In order to navigate the ontology graph, we define the following primitive operations: $RChild$, $ANCES$, $DESC$, and $SYNs$. We need these operations to determine concepts and relationships that are of interest for query processing as we shall see in Chapter 5.

Let $P_{ths}(c_i - c_j)$ be a set of paths connecting two concept nodes c_i and c_j. Let be $c_1, c_2, s_k, s_h \in \zeta$ and $\beta, \beta_i \in \Re$:

- $Rchild(\beta, c_1) = \{c_2 \mid G(c_2, \beta, c_1)\}$
- $DESC(\beta, c_1) = \{c_2 \in \zeta \mid \exists\, p \in P_{ths}(c_2 - c_1) : \forall\, e = (s_k \beta_i s_h) \in p\ ,\ \beta_i = \beta\}$
- $ANCES(\beta, c_1) = \{c_2 \in \zeta \mid \exists\, p \in P_{ths}(c_1 - c_2) : \forall\, e = (s_k \beta_i s_h) \in p\ ,\ \beta_i = \beta\}$
- $SYNs(c_1) = \{c_2 \mid G(c_1,\ \texttt{"SynOf"}, c_2)\}$

Informally, $Rchild(\beta, c)$ is the set of all descendant concepts of c following edges of type β. $DESC(\beta, c)$ returns the set of all descendant concepts of c following edges of type β, whereas $ANCES(\beta, c)$ returns the set of all ascendant concepts of c by following edges of type β. Similarly, $SYNs(c)$ returns the set of all synonyms of c.

3.7 Problem Definition for this Thesis

Need of Background Semantics. Traditional database management systems, which only rely on syntactic approaches, might not answer user queries effectively. This is due to problems caused by the lack of data semantics during query processing. The major problems are: (i) hiding the semantic heterogeneity of integrated data from the users, (ii) overcoming semantic confusion in terminologies caused by employing synonyms and homonyms, and (iii) providing users with the most relevant answers to his requests in less time and/or resources. There is a general agreement that such semantic problems could be achieved only with a good understanding of the semantics of the database content. However, most of these semantics are not explicitly represented in database systems. They are embedded at various levels (e.g., in the database model, in data structures, in constraints on the data, and in data domains) and not explicitly stated, yet well understood by the user community. In addition, semantics of real-world entities represented in the database may not be completely described. In fact, during the design and maintenance phases of a database some domain semantics might not be captured or else be captured but removed due to representation limitations of database management systems. In fact, during the process of database design, the mapping from the conceptual schema to the physical schema induces a progressive degradation of semantics often with loss of most of the information about involved concepts and relationships. Furthermore, a database may probably grow within the organization over many years and might have its definitions modified by different designers. The documentation is often out of date or non-existent, especially with old systems, so that it becomes difficult to know exactly what the data and the relationships between the data really mean. In this context, current research in database management becomes aware of capturing additional semantic knowledge from other background sources to deal with the problems above.

Semantic Query Transformation. This thesis addresses the problem of how to improve answers of database inquires using additional information in form of semantic knowledge. Given a database and a set of semantic knowledge, which capture meaning of its content, we will show how these knowledge could be exploited effectively to reformulate a user query such that the new query can provide more "meaningful" answers meeting the intention of the user. Our goal is to allow a DBMS to deal with user queries both at the semantic as well as the syntactic level. There, users do not need to fully understand the database content to issue their queries and the resulting database answers could fulfill their expectations completely. Previous approaches of query optimization, as outlined in Section 2.2.2, require that the transformed queries must be equivalent to the original ones. We argue that this requirement should be relaxed in many situations where a user does not need only "exact" answers for their queries but those answers that meet his intention. We define an exact answer as being the set of database items which literally matches the terms of query predicates. We will present how a query can be transformed into a new one which is not necessary equivalent but can provide more meaningful answers. In fact, when a user queries a database in order to retrieve information about certain objects, the answer to his query might not satisfy his needs. This can be justified by several facts. First, the information stored in the database is usually captured in natural languages. This leads to several variations in expression of the same concept (synonym problem). Moreover, languages introduce multiple meanings of the same expression (homonym problem). These problems might affect query results when formulating queries using certain terms. Second, if a user wants to retrieve information from a database about certain entities, he might use terms which do not exactly match database values (due to the mismatch between the user's world view and the database designer's world view). However, there might be values in the database that are syntactically different from one another but semantically equivalent to the user terms and that express the same intention of the user. We define two sets of terms to be semantically equivalent if they have the same meaning, i.e., if their related concepts and relationships in the ontology identify the same real-world entities. For example, if two terms are synonyms, they are semantically equivalent. Thus, there might be different ways to formulate a query using semantically equivalent terms. We address this issue as a semantic problem rather than as a pattern matching problem. Third, some results in the answer might not be related to the same context associated with the query. The context must be defined by the user. Finally, a user may need additional information or even different information than the query requests [HHCF96]. Furthermore, he may prefer an alternative answer to his queries over not receiving any answer at all [LNR97]. We consider the following example to illustrate these ideas.

Example 3.1 Assuming we have a relational database, denoted by `PDB`. This database contains information about technical items of a store and includes two relations called `Item1` and `Component`: the relation `Item1` contains a set of items described by the attributes `name`, `model`, and `price`. The relation `Component` contains the parts belonging to each item. The relational schema of `PDB` is described as follows:

Item1 (AID, Name, Model, Price)
AID: Item identifier
Name: Name of the Item
Model: Model of the Item
Price: Price of the Item
PKey(AID)

Component (SID, MID)
MID: Main part identifier
SID: Second part identifier
FKey(MID) TO Item1
FKey(SID) TO Item1
PKey(SID,MID)

Suppose, at present, that PDB contains the instances as shown in the Tables 3.1 and 3.2.

AID	Name	Model	Price
123	computer	ibm	3000 $
124	intelPc	toshiba	5000 $
125	notebook	dell	4000 $
127	pc	compaq	2500 $
128	product	hp	3000 $
129	monitor	elsa	1000 $
135	keyboard	itt	80 $
136	desktop	ibm	1000 $
140	macPc	mac	2000 $
141	calculator	siemens	1500 $

Table 3.1: Item1 relation

SID	MID
123	129
123	135
123	136
124	129
124	135
124	136
125	135
127	129
127	135
127	136
128	129
128	135
128	136
140	129
140	135
140	136
141	135

Table 3.2: Component relation

Querying the database PDB to retrieve information about the Item "computer" also means information about the Items "data processor" and "calculator" because these terms are synonymous with the term "computer". Consequently, if a user formulates his query specifying only the term "computer" he might miss other tuples concerning "data processor" and "calculator". In addition "computer" is implied by other terms which should be considered in the query. We deliberately chose a simple example here for illustration purposes, but there are more complicated ones depending on the nature of the query as we shall see later. In fact, the difference between the user's perception of real world objects and the view of the database designer, who registers information about these objects, might cause semantic ambiguities including the "vocabulary problem". Therefore, it is hard for the DBMS to solve such semantic problem without additional knowledge. To cope with this problem we propose to use an ontology as a source of semantic information describing the underlying database. We assume that users know about the structure of the database but not about its content. □

In summary, we state our problem as follows:
Given a database DB, an ontology O and a user query Q, find a reformulated query Q' of Q by using O such that Q' returns a more meaningful answer to the user than Q.

3.8 Summary

The goal of this chapter is to illustrate the importance of understanding the meaning of data in order to efficiently and effectively use them in scientific applications. Data are represented by symbols (e.g., terms). We explicated the relations between meaning, symbols and things they refer to. Users have different interpretation of terms, they use different jargon and they may denote overlapped or mismatched concepts regarding the others. The resulting lack of a clear understanding of the meaning causes many problems for the user's community. We suggest ontologies as a basis source for capturing explicit semantics. We have taken cues from philosophy and artificial intelligence to define and represent ontologies. Furthermore, we have shown their effective usefulness in specifying and explaining data sources. In particular, the use of ontologies is a promising approach in order to facilitate querying heterogeneous sources.

We also discussed the issues related to database models and pointed out the differences between the two. The main distinction criteria is that database models are based on application requirements but ontologies are created based on the understanding of real-world entities with minimal commitments of the community members. Ontologies complement semantics embedded in database schemas that represented only a subset of the domain of interest. Using additional semantics from an underlying ontology might enhance query answers. We defined the problem of how to use ontologies for transforming user queries in order to obtain more meaningful answers. The transformation should take semantic information about both schema structure and instances of a database into account. The solution of this problem is presented in the next chapters.

Chapter 4

Query Processing Using Ontologies

The main contribution of this thesis is to bring together the concept of query processing in databases and the concept of semantic knowledge. Having introduced them in Chapters 1 and 2, respectively, we combine the two in this chapter to introduce an approach to query processing using semantic knowledge.

Ontologies have gained popularity in the fields of database and information systems as a good support for many applications like data integration, data mining, data annotation, and searching web documents. The key point in databases is the ability to make their semantics available, i.e, they could express the meaning of their structure and content. There, problems caused by data heterogeneity, for example, could be alleviated using knowledge from ontologies. In Section 4.1, we present these problems in more detail. Next, we describe how ontologies are used for supporting data management. Indeed, for an effective use of ontologies we need to map ontologies to databases. In Section 4.3, we specify the different types of mappings at the instance level and at the structural level as well. Then, we show in the subsequent section how these mapping can be established. Finally, we review some related work.

4.1 Semantic Heterogeneity

In data integration systems, data are unified according to some common semantics but are located in different information sources. *Semantic heterogeneity* is one of the most crucial problems for such systems. Semantic heterogeneity is concerned with the meaning of data in user requests as well as in information sources. In fact, information sources always abstract the real world according to the services they are providing. They neither view nor represent all details of the real world. Therefore, any source has its own semantics and use different terminologies to describe entities of the real world. Different representations and interpretations of data cause *semantic conflicts* (also known as incompatibilities) when they are exchanged or retrieved from sources. There are several classifications of semantic conflicts found in the literature [KS91, SK93, Zha92, CW93]. For instance, semantic conflicts can be classified at two levels of representation: the *data level* and the *schema level* [KLK91].

The first class, called *data conflicts*, covers those conflicts that arise due to the inconsistent representations of data across information sources. There are three types of

such conflicts: *expression*, *unit*, and *precision* conflicts [KLK91]. Expression conflicts occur when different expressions are used for same data or one expression is used for different data. For instance, multiple strings might be used to represent the same date (e.g., "12-July-05", "12/07/05"). The word "apple" may be used to denote a fruit in a database whereas it may be used to denote a computer in another database. Unit conflicts occur when different units are used for the same numeric data. For instance, a database provides flight distances in miles while another database provides these distances in kilometers. Conflicts in precision arise when different information sources represent the same data with various levels of precision. For instance, two databases may use values from a domain of different cardinalities. One database may report building grades using four-point scale defined as strings (e.g., "excellent", "good", "acceptable", "unsatisfiable") while another database may report the grades using five-point scale as integers (e.g., 1, 2, 3, 4, 5).

The second class, called *schema conflicts*, covers those conflicts that arise when different sources use different schema definitions for representing the same real-world entities of the same application domain. For example, a database may represent employee's departments as an attribute while others may represent them as a relation. In addition, schema conflicts result from the use of different specifications for the the same schema definition. For example, a database may define a person as "male" and "female" while others as "student" and "professor". Schema conflicts include *naming conflicts*, *schema-isomorph conflicts*, *generalization conflicts*, and *aggregation conflicts* [RS92]. Naming conflicts occur when different names are arbitrary assigned to schema elements (e.g., relation names, attribute names). There are two types of such conflicts: (i) the same name is used to specify semantically different elements and (ii) different names are used to specify semantically equivalent elements. Two elements are considered semantically equivalent if they describe the same entities. For example, two attributes labeled as "References" in one database and "Publications" in another database may represent the same research papers. Furthermore, an attribute labeled as "Document" may be used for books in one database while it may be used for papers in another database. Schema-isomorph conflicts occur when different sources represent different properties of the same set of entities. That is, in relational databases, two relation schemas describing the same entities have different number of attributes. Generalization conflicts are caused by representing a set of entities at different level of generalization in different sources. For instance, PhD students may be represented by a relation `PhD-Student` in a database whereas they may be represented by a relation `Student` which describes all students in another database. Aggregation conflicts arise when an aggregation is used in a source to identify a set of entities in another source. That is, aggregation may involve some properties of entities in one source to specify the properties of other entities in another source.

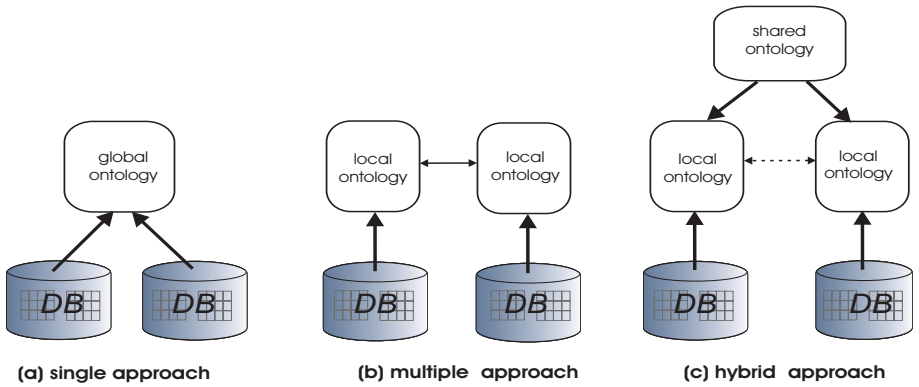


Figure 4.1: Three approaches to integrating Sources using Ontologies

4.2 Use of Ontologies for Query Processing

4.2.1 Basis for Mediation

The problem of semantic heterogeneity has been generally addressed by two types of approaches: approaches without using ontologies (e.g., metadata [She91]) and approaches using ontologies. The approaches of the first type attempt to define a federated (global) schema over the local schemas of the sources. The approaches of the second type use an ontology as an intermediate layer between multiple sources and human users or application programs [Wie94]. The clear descriptions of vocabularies in ontologies can reduce semantic ambiguities and hence unify different sources in a common semantic view. Thus, applications that share common vocabularies can easily communicate with each other in order to exchange information. To this end, ontologies can be employed within three architectures as shown in Figure 4.1. In the first architecture one single ontology is used for different sources (see Figure 4.1.a). In this case sources provide nearly the same view on the domain. Each source should use the same vocabulary from a global ontology as other sources. A practical example is the TAMBIS ontology [PSB+99] which acts as a global ontology describing different biological sources. Items in different data sources are annotated with concepts from this ontology in order to enhance understanding of their meaning. However, if each information source has its own semantics and defines terms with respect to only its own perspective then committing to a single ontology becomes very hard (if not impossible). One solution to this problem is to describe each source by a separate ontology (local ontology) and define mappings between them as shown in Figure 4.1.b. These inter-relationships specify semantics between terms used in different local ontologies, e.g., which terms have the same meaning. A practical example using this approach is the OBSERVER system [MKSI96]. However, from a practical viewpoint mappings between the source ontologies are hard to define and querying the underlying sources becomes very difficult. To overcome these shortcomings a hybrid architecture could be an appropriate alternative (see Figure 4.1.c). A global ontology is build over local ontologies. It provides vocabularies of higher granularity in order to describe complex relationships between the terms of the local ontologies. The drawback of this approach is the difficult main-

tenance of ontologies when local ontologies are modified [WVV+01]. We review some systems which implement the above approaches in Section 4.5.

4.2.2 Query Enhancement

Ontologies can improve the capability of query processing in both information retrieval and database management systems.

In the information retrieval field, searching for a specific information across multiple Web-sources, which are probably written in different languages usually results in a huge number of results. Most of these results are irrelevant to the user. The major reason of this problem is that conventional information retrieval techniques either rely on simple keyword-based analysis of user queries or on a specific encoding of the information. The former approach has the drawback that input terms of the user might not be completely consistent with the terms of the source. For instance, the same word can be used in different ways, i.e., it may be used as verb, adjective, or noun. The English word "repair" might denote a kind of activity (putting something to work again) if it is used as noun or it might denote an action (set right) if it is used as verb. The latter approach may reduce the recall of queries because relevant information with slightly different encoding could not be matched. On the other hand, keyword-based search might reduce precision of the information because matched words might be ambiguous. The use of ontologies can help to overcome the limitations of these approaches. An ontology that provides vocabularies for classifications will improve the search process through, e.g., expanding queries using words from the ontology which have the same meaning. These kinds of improvement increase the recall of queries since closely related results could be retrieved. The use of ontologies containing formal representation of semantics (e.g., Ontolingua [FFR96]) enables reasoning about relationships of query terms. This can be done, for example, by matching more specific terms and hence might increase recall and precision.

Currently, many efforts have been devoted to realize the Semantic Web vision [BLHL01]: embedding semantic knowledge in Web sources in order to enable information retrieval in unambiguous and automatic ways. For instance, Web documents could be annotated from ontologies, i.e., documents are indexed with ontological terms and concepts instead of simple keywords. This approach has several advantages. First, ontological abstractions provide a degree of independence from changes that may occur in the document. The content might change using equivalent terms, e.g., "ontological commitments" instead of "commitments on ontology". However, if we enrich the document with ontological terms, the search result may not provide irrelevant information. Second, since the ontology used for enriching documents might be domain specific, the interpretations of keywords are bound to that domain and therefore document retrieval is likely to be more efficient. A term can have several meanings in different domains. By first mapping the keyword to its semantic representation in a specific ontology and then using the ontology's concepts and relationships, a much more focused search approach can be taken. The document specific representations no longer affect the search. This is extremely important in the case of multilingual representations. Keywords of several languages are mapped to the same concept in an ontology and are therefore given the

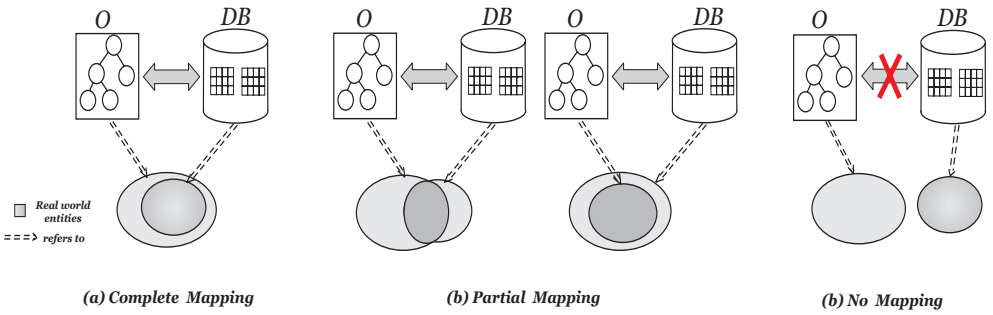


Figure 4.2: Mapping cases

same meaning. For instance, portals for multilingual search can be set up to return the same results independant from the language used for the retrieval [Lau03].

In database management systems, ontologies can play the role of a query model for processing user queries. As we have seen in the last subsection, ontologies provide promising solutions for data integration problems. In data integration, ontologies can be used as a global query schema over multiple databases. Users formulate their queries over the ontology without directly accessing the database. That is, they do not need to know how data are structured in databases. However, they should be familiar with the content of the ontology. Queries are formulated in terms of the ontology vocabularies. Using mapping information between ontology and underlying databases the global query will be translated into sub-queries for each database and different results will be combined for the answer. Indeed, ontologies allow for querying databases based on concise semantics which can be intuitive to users of the same domain. Moreover, they allow to identify semantic links between different sources using relationships between their concepts in the ontology. This approach was implemented in several systems, e.g., SIMS [AHK01], Infosleuth [BBB+97], and TAMBIS [PSB+99] as described in Section 4.5.

In this thesis, we use ontologies for query reformulations within single database systems. We exploit semantics from an ontology describing a given database in order to enhance query answers. Thus, answers will be more aware of the user intention, hence better satisfy his needs.

4.3 Mappings between Ontologies and Databases

The task of involving ontologies in querying databases requires a clear specification of how ontology components (concepts and relationships) are linked to database components (relations, attributes, and data values). This *mapping* specifies for certain components in the database the corresponding components in the ontology that describes their intended meaning. Our definition of a mapping is based on how an ontology and a database model the same real world entities. More clearly, given an ontology O and a database DB, we distinguish between three cases of mappings: *complete*, *partial*, and *non existent*. Figure 4.2 graphically illustrates these cases. The dashed arrows denote the correspondences of an ontology/database to the real world entities. A mapping is

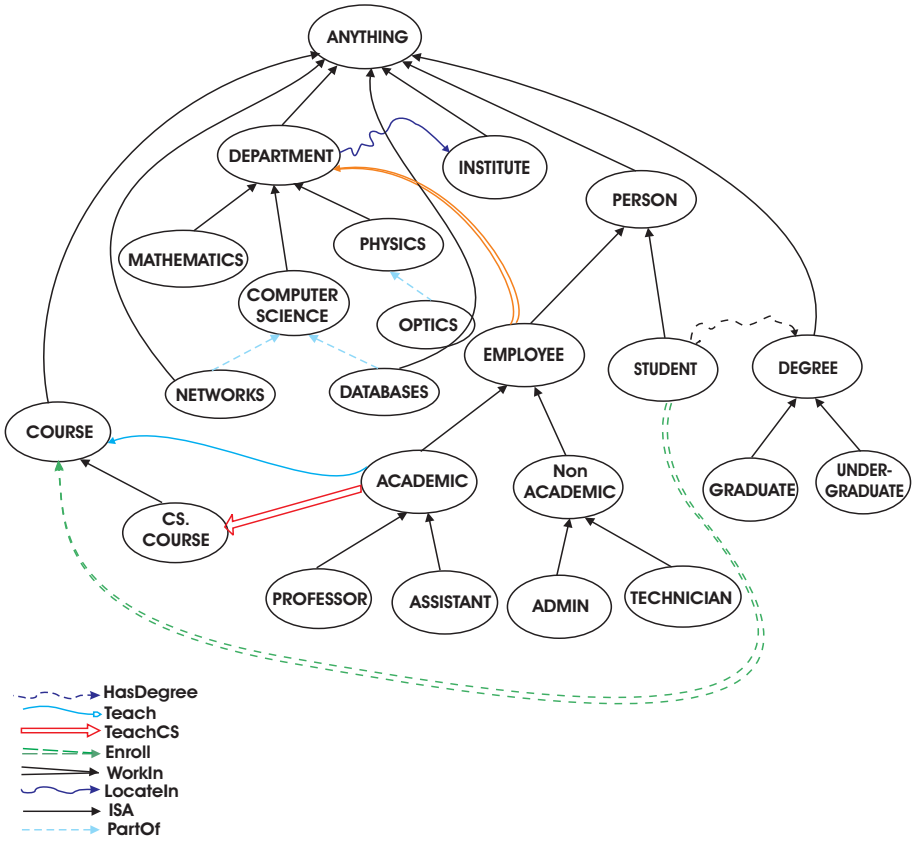


Figure 4.3: Employee Ontology (EmO)

complete if the set of entities represented by concepts in O includes all entities modelled in DB. That is, O completely describes the entities represented by DB. For instance, an ontology describing persons could be mapped to a database representing employees since a given employee is a person. However, if O covers only a part of the domain of DB, then the mapping is *partial*. For instance, DB may represent instances of cars and trucks of a company, but O may only describe cars. Finally, if the domains concerned by O and DB are disjoint, then the mapping is inexistent. For instance, O may describe animals while DB may represent cars.

Given an ontology structure $O = \{G, \zeta, \Re, \Im\}$ and a database DB having a schema $S = (\Sigma, I)$ and an extension $Ext(DB)$. We define in the next sections the different types of mappings in more detail and use the university database UNI-DB (see Example 2.1) and the ontology depicted in Figure 4.3 to present some illustrative examples. Due to the lack of space we present in Figure 4.4 only three relations of UNI-DB database.

4.3.1 Correspondences between Ontology Elements and Database Extension

In this section, we define the correspondences that might exist between elements in an ontology and instances in the underlying database (database extension). That is, how ontology elements (concepts and relationships) correspond to database instances

SSNr	FName	LName	Position	DNr
SN02	John	Been	Professor	01
SN03	Smith	Gray	Assistant	02
SN04	Smith	Drake	Technician	03
SN05	Patrick	Clay	Professor	04
SN06	Dick	Sheen	Staff	05
SN07	Maria	Moore	Technician	01
SN08	Jane	Clemens	Academic	03
SN09	David	Larson	Prof.	03

(a) Employee

DNr	DName	Institute
01	CS.	HUI
02	Math.	FUI
03	Elec.	TUI
04	Phys.	HUI

(b) Department

UNr	DNr	UName
01	01	Databases
02	01	Networks
03	02	Optics

(c) Division

Figure 4.4: UNI-DB relations

(tuples, single values). Formally, we define this correspondence as a set of 4-tuples $< e, Exp, Rlist, Alist >$ where e is an element in the ontology and Exp is a declarative expression which represents the *exact* set of database instances to which e matches. *Rlist* and *Alist* provide the names of relations and attributes that are involved in the correspondence, respectively. We refer to them as *arguments* of a correspondence. We identify three types of such correspondence:

1. Correspondences between concepts and attribute values.
2. Correspondences between concepts and relations.
3. Correspondences between relationships and relations.

Correspondences between concepts and attribute values

As we introduced in Section 3.3, labels for concepts are in principle words from some natural language. In natural languages words may be denoted in different ways using, for example, several acronyms and abbreviations. For instance, the word "computer science" can be shortly written as "CS.". Therefore, a correspondence σ_i between a concept c_i in ζ and a value v_i in the attribute domains means that the concept name and the value are semantically equivalent.

Formally, σ_i is a pair $< c_i, v_i >$ such that there exists an equivalence relationship between c_i and v_i like Acronym and Abbreviation. We refer to this kind of correspondence as *data correspondence*.

Definition 4.1 (Concept-Value Mapping) *A mapping between concepts in ζ and database values D is a relation $\Psi_{\zeta\mathcal{D}} \subseteq \zeta \times D$ such that $\Psi_{\zeta\mathcal{D}} = \{(c_i, v_i)\}$, where $c_i \in \zeta$ and $v_i \in D$ are arguments of data correspondences σ_i.*

Given an attribute A_i, we denote by $\Psi_{\zeta\mathcal{D}}^{A_i}$ the mapping between values of the domain of A_i ($dom(A_i)$) and concepts in ζ. The Concept-Value mapping is then a set of data correspondences between each pair of them. This mapping is of type many-to-many mapping. That is, each database value might be matched to a single or multiple concepts and a given concept might be matched to one or more values. For instance, if the relation `Employee` in the `UNI-DB` database contains the instances as described below, the concept `PROFESSOR` in the ontology `EmO` could be then mapped to the values {"$Prof.$","$Professor$"} of the attribute `Position` in the relation `Employee`. However, if a value has multiple homonyms, such as the term "star", it might be mapped to multiple concepts.

Correspondences between concepts and relations

We recall that concepts represent a group of *concrete* or *abstract* entities. Let Ent denotes the set of all entities of an application domain. Each concept can be associated with a subset of Ent, i.e., there exists an interpretation $I : \zeta \rightarrow 2^{Ent}$ that associates each concept name with their entities. Each entity is instantiated by a data value in the database. However, a value could denote different entities. For instance, "Smith" could be the name of two persons. To map a given concept to its corresponding instances in the database, the instance values should be identified in a unique manner, i.e., each value matches only one entity. In database context, this identification can be achieved using key values (e.g., primary keys). In this section, we attempt to define the semantic of matching concepts with their corresponding entities based on the correspondences between concepts and the set of values representing these entities in the database.

Definition 4.2 (Concept Correspondence) *A* correspondence ψ_1 *between a concept* $c \in \zeta$ *and a set of database instances (tuples) in* $Ext(DB)$ *is defined as a 4-tuple of the form:* $\psi_1 =< c, Exp, Alist, Rlist >$ *where Exp is a domain relational calculus (DRC) expression defined on a set of relation schemas Rlist including a set of attributes Alist. Exp specifies a (derived) relation containing only key values which correspond to the entities of* `c`. *We refer to this kind of correspondence as* concept correspondence.

Based on this definition we can define a mapping between ζ and database extension E as a set of all correspondences between concepts in ζ and DRC-expressions representing subsets of E.

Definition 4.3 (Concept-Extension Mapping) *A Concept-Extension mapping is a defined as a relation* $\Psi_{\Re\mathcal{E}} = \{(c_i, Exp_i)\}$, *where* c_i *and* Exp_i *are concepts and DRC-expressions derived from concept correspondences* [1].

Types of concept correspondences: As shown in Figure 4.5 we distinguish three types of correspondences:

1. A correspondence between a concept and a (derived) relation.
2. A correspondence between a concept and a set of relations.
3. A correspondence between a concept and a relation extension.

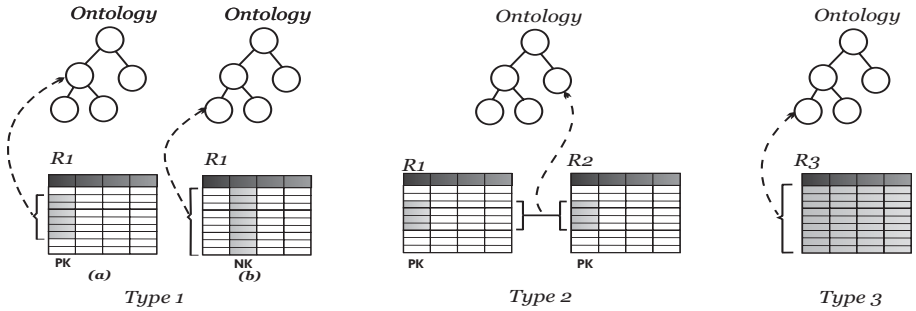


Figure 4.5: Concept Correspondences

The first type of correspondence associates a given concept with a (derived) relation. More precisely, this correspondence associates a concept with a set of tuples from a relation extension which satisfy certain conditions. This set contains then all database values that are related to a concept. On one hand, there is a correspondence for concrete concepts [2] (see Figure 4.5 (type 1.a)). For example, the concept `TECHNICIAN` can be mapped to only a subset of the extension of the relation `Employee` that is related to technicians. This set contains tuples which have 'technician' as value for the attribute `Position`. Formally, this mapping correspondence is defined as $< TECHNICIAN, Exp_{1a}, \{SSNr\}, \{Employee\} >$ where $Exp_{1a} = \{x \mid \exists\, y\, z\, u\, w\ Employee(x, y, z, u, w) \wedge u = \text{"}technician\text{"}\}$. In this case the attribute list contains only primary keys (PK). On the other hand, there is a correspondence for abstract concepts [3] (see Figure 4.5 (type 1.b)). There, the attribute list contains non key attributes (NK). An example of such correspondence is $< INSTITUTE, Exp_{1b}, \{Institute\}, \{Department\} >$ where $Exp_{1b} = \{y \mid \exists\, x\, z\ Department(x, y, z)\}$.

The second type of correspondence associates a given concept with more than one relation. For instance, in the ontology `EmO`, the concept `PERSON` could be mapped to a set of tuples from the relations `Employee` and `Student`. Formally, this mapping correspondence is defined as $< PERSON,\ Exp_2, \{SSNr, INr\}, \{Employee, Student\} >$, where $Exp_2 = \{x \mid \exists\, y\, z\, u\, w\ Employee(x, y, z, u, w) \vee Student(x, y, z, u)\}$.

The third type of correspondence associates a concept with the whole extension of a relation. In this case a concept and its neighborhood concepts [4] are mapped to instances of the same relation schema and all attributes participate in this mapping. For instance, the concept `DEPARTMENT` can be mapped to the extension of the relation `Department` since its neighborhood concept `INSTITUTE` can also be mapped to the same relation. This mapping correspondence is defined as $< DEPARTMENT, Exp_3, \{DNr, Institute\}, \{Department\} >$, where $Exp_3 = \{(x, y) \mid \exists\, z\ Department(x, y, z)\}$.

[1] Omit *Alist* and *Rlist* from ψ_1 expression

[2] Concepts which represent concrete entities in the world

[3] Concepts which represent abstract entities in the world

[4] Set of concepts that are related directly to a given one.

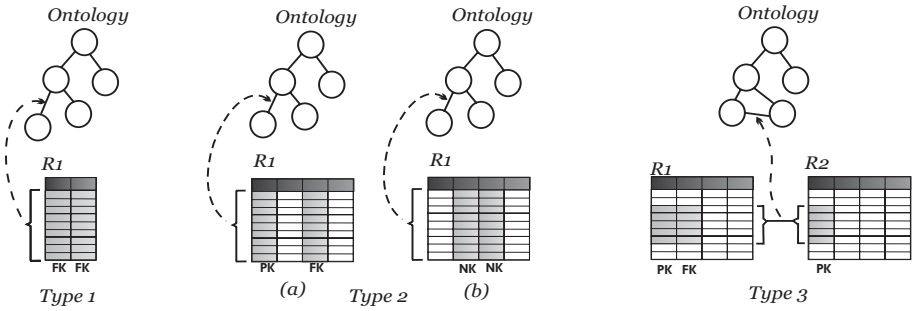


Figure 4.6: Relationship Correspondences

Correspondences between relationships and relations

Definition 4.4 (Relationship Correspondence) *A* correspondence ψ_2 *between a relationship* $\beta \in \Re$ *and a set of database instances is defined as a 4-tuple:* $\psi_2 =< \beta, Exp, Alist, Rlist >$*, where Exp is a domain relational calculus (DRC) expression defined on a set of relation schemas Rlist including a set of attributes Alist. Exp specifies a (derived) relation containing only key values which correspond to the instances that are related through* β*. This kind of correspondence is called* relationship correspondence.

Based on this definition we can define a mapping between $\Re$ and database extension E as a set of all correspondences between relationships in $\Re$ and (DRC) expressions representing subsets of E.

Definition 4.5 (Relationship-Extension Mapping) *A Relationship-Extension mapping is defined as a relation* $\Psi_{\Re\mathcal{E}} = \{(\beta_i, Exp_i)\}$*, where* β_i *and* Exp_i *are derived from Relationship Correspondences* [5]*.*

Types of relationship correspondence: We can distinguish, as shown graphically in Figure 4.6, three types of correspondences:

1. A correspondence between a relationship and a relation extension.

2. A correspondence between a relationship and a (derived) relation.

3. A correspondence between a relationship and a set of relations.

At the semantic level, we note that if a relationship between a concept c_1 and a concept c_2 exists, then each entity of c_1 is related to an entity of c_2 through that relationship. We say that c_1 and c_2 support that relationship. At the database level, a relationship is interpreted as the information that link pairs of key values representing these entities. Therefore, mappings of relationships depend also on mappings of the concepts they relate, as described below.

The first type of correspondence associates a relationship with all tuples of a relation extension. In this case the concepts supporting such a relationship are mapped

[5]Omit *Alist* and *Rlist* from ψ_2 expression

to relations belonging to the same relation schema. One of the main features of this correspondence is that the relevant relation schema contains exactly two key attributes. For instance, the relationship `Enroll` in the ontology `EmO` relates the concept `EMPLOYEE` with the concept `COURSE`. This relationship could be mapped to the extension of the relation `Enrollment` implying that for each pair of entities of `STUDENT` and `COURSE` there exists a tuple in the relation `Enrollment` which asserts their relationship. Formally, this correspondence could be formulated as $< Enroll, Exp_4, \{INr, CID\}, \{Enrollment\} >$ where $Exp_4 = \{(x, y) \mid \exists\, y\, Enrollment(x, y)\}$.

The second type of correspondence denotes an association between a relationship in an ontology and a set of tuples in a relation. Here, we distinguish between two cases. In the first case one of the concepts that support the relationship is mapped to values from the tuple set (see Figure 4.6 (type 2.a)). For instance, the relationship `WorkIn` can be mapped to tuples in the relation `Employee` which are denoted by $Exp_5 = \{(x, v) \mid \exists\ y\, z\, u\, Employee(x, y, z, u, v)\}$. Formally, the correspondence of `WorkIn` is formulated as $< WorkIn, Exp_5, \{SSN, DNr\}, \{Employee\} >$. In the second case, the concepts supporting the relationship are mapped to values of two non-key attributes as shown in Figure 4.6 (type 2.b).

The third type of correspondence associates a relationship with a set of tuples derived from more than one relation (see Figure 4.6 (type 3)). For instance, the relationship `TeachCS` in `EmO` relates the concept `ACADEMIC` with the concept `COURSE`. It describes the academic employees who teach computer science courses. This relationship can be mapped to a set of tuples derived from the relations `Lecturer` and `CS-Courses`, namely to the result of the join between those relations. Formally, this correspondence has the form: $< TeachCS, \{LNr, CID\}$, {Lecturer,CS-Courses}, $Exp_6 >$, where $Exp_6 = \{(x, y) \mid Lecturer(x, y) \wedge \exists\, w\, z\, [$ CS-Courses(w,z) $\wedge\, y = w]\} >$.

4.3.2 Mappings between Ontologies and Database Schemas

The mappings between Ontologies and Database schemas determine how concepts and relationships in an ontology are related to relations and attributes in the underlying database schema. We define a mapping as a set of *mapping elements* which has the following form: $< e, \{e'_1, \ldots, e'_n\} >$ where $e \in \zeta \cup \Re$ and $\{e'_1, \ldots, e'_n\}$ is a set of attribute names, relation names, or database values.

In this section, we define semantics of the mappings based on the above correspondences. Therefore, we distinguish four types of mappings:

1. Mapping between concepts and single attributes.
2. Mapping between concepts and relations.
3. Mapping between relationships and attributes.
4. Mapping between relationships and pairs of attributes.
5. Mapping between relationships and relations.

Moreover, we attempt to formalize these mappings as mathematical relations and denote them by predicate symbols for expressing some assumptions. For the remaining of this thesis, we use the same name for a predicate and its relation.

Mapping between concepts and single attributes

A mapping between concepts from ζ and attributes from U can be defined as a set of mapping elements of the form: $(c_i, \{A_1, \ldots A_k\})$. Therefore, the mapping is of type a one-to-many mapping. That is, each concept in ζ might cover one or more attributes of a schema. For example, suppose we have a concept `NAME` in the ontology `EmO` and is connected to the concept `EMPLOYEE` to denote that an employee has a name. Thus, the concept `NAME` could be mapped to both attributes `FName` and `LName` of the relation employee to indicate that an employee name is obtained by concatenating the first name with the last name. In this thesis, we focus on one-to-one mapping between concepts and attribute names.

The Concept-Attribute mapping is based on the concept correspondences of type 1.b as in Figure 4.5. It is a set of pairs (concept, attribute) derived from expressions of such correspondence.

Definition 4.6 (Concept-Attribute Mapping) *A mapping between a concept set ζ and a set of database attributes U is a relation $\Psi_{\zeta\mathcal{U}} \subseteq \zeta \times U$, $\Psi_{\zeta\mathcal{U}} = \{(c_i, A_i)\}$ such that $c_i \in \zeta$ and $A_i \in U$ are arguments of concept correspondences.*

There are two different interpretations of this mapping. First, a mapping between a concept and an attribute means that each value of the attribute represents an entity of that concept. For instance, the concept `INSTITUTE` in ontology `EmO` could be mapped to the attribute `Institute` of the relation `Department` indicating that each `Institute` value represents an entity of `INSTITUTE`. Second, the mapping means that each value of the attribute represents an entity of one of its sub-concepts. For instance, consider the extension of the relation `Student` as shown in Figure 4.4. The concept `EMPLOYEE` in `EmO` could be mapped to the attribute `Position` whose values are mapped to the sub-concepts `PROFESSOR`, `TECHNICIAN`, `ACADEMIC`, etc.. We denote such mapping as Ψ^*_{CA}. In this thesis we assume that all values of the domain of A_i are mapped to sub-concepts of the concept representing A_i. However, real-world entities modelled in a relation extension are in fact represented by only leaf concepts [6] (we recall that all concepts are disjoint). For instance, the employee 'SS08' has a position 'academic' but intuitively this means that he should be either a professor or an assistant. Formally, these assumptions are expressed as follows [7].

Given a relation $R(A_1, \ldots, A_n)$, where A_1 is its primary key, $D_i = dom(A_i)$ $(i = 2 \ldots n)$ and $C_0 \in \zeta$.

[6] A concept which does not have any sub-concept is called a leaf concept

[7] Leaf(c,l) is a predicate denoting a relation between a concept c and its leaf concept l

Sub-concept Constraints [8]

H_1: $\forall\, v_2 \in D_i :\ \ \Psi^*_{CA}(C_0, A_i) \implies \exists\, c \in \zeta\ \ \Psi_{\zeta\mathcal{D}}(c, v_2) \wedge Isa(c, C_0)$

H_2: $\forall\, (v_1, \ldots, v_n) \in D^m$
$\Psi^*_{CA}(C_0, A_i) \implies \exists\ c\ l\ \in\ \zeta\ \ R(v_1, \ldots, v_n) \wedge \Psi_{\zeta\mathcal{D}}(c, v_2) \wedge \Psi_{\zeta\mathcal{D}}(l, v_1) \wedge$
$Leaf(c, l)$

Mapping between concepts and relations

A mapping between a set of ontological concepts c_i from ζ and relation names R_i from Σ associates each relation (or group of relations) with a concept that describes all entities represented in its schema (their schemas). This mapping can be defined as a set of mapping elements of the form: $(c_i, \{R_1 \ldots R_k\})$. An element of this mapping can be determined based on concept correspondences of type 1.a, type 2, or type 3 as in Figure 4.5. We restrict ourselves to a one-to-one mapping. That is, a concept can be only mapped to one relation name and vice versa.

Definition 4.7 (Concept-Relation Mapping) *A mapping between a concept set ζ and a set of relation names Σ is a relation $\Psi_{\zeta\Sigma} \subseteq \zeta \times \Sigma$ such that $\Psi_{\zeta\Sigma} = \{(c_i, R_i)\}$, where $c_i \in \zeta$ and $R_i \in \Sigma$ are arguments in concept correspondences.*

This mapping also means that if there are multiple concepts belonging to the same `ISA`-hierarchy and being candidates for the mapping then the least general of them should be considered. For example, the concept `STUDENT` in `EmO` will be mapped to `Student` in `UNI-DB`. However, the concept `ANYTHING` or `PERSON` will not be mapped to `Student` because they have more general meaning than `STUDENT`. In addition, we shall mention that we consider only the concept correspondences that match all tuples of a relation extension. Otherwise, a concept could not cover all knowledge about the entities represented in the relation. For instance, a mapping element such as (`PROFESSOR`, `Employee`) will be ignored.

In addition, we denote a Concept-Relation mapping as $\Psi^*_{\zeta\Sigma}$ if for each element (C_R, R) of the mapping there is an attribute A $(A \in R)$ which is mapped to a concept C_A and is a sub-concept of C_R. Formally, $\Psi^*_{\zeta\Sigma}(C_R, R) \implies \Psi^*_{\zeta\mathcal{U}}(C_A, A) \wedge Isa(C_A, C_R)$. An example of such mapping element is (`EMPLOYEE`, *Employee*).

Mapping between relationships and attributes

A mapping between the set of ontological relationships $\Re$ and the attribute set U associates each foreign key attribute of a given relation schema with a relationship in O. Like $\Psi_{\zeta\mathcal{U}}$ we address the mapping of type one-to-one, i.e., one concept name with one attribute name.

Definition 4.8 (Relationship-ForeignKey Mapping) *A mapping between relationships $\beta_i \in \Re$ and foreign key attributes $K_i \in U$ is defined as a relation $\Psi_{\Re\mathcal{F}} \subseteq \zeta \times U$ such that $\Psi_{\Re\mathcal{F}} = \{(\beta_i, K_i)\}$, where β_i and K_i are arguments in relationship correspondences.*

[8] We denote by D^m the cartesian product of all attribute domains

This mapping specifies the kind of relation that exists among the entities represented in the database schema. Therefore, an element of such mapping can be determined from a relationship correspondence of type 2.a as in Figure 4.6. For instance, the relationship `WorkIn` in `EmO` can be mapped to the attribute `DNr` in the `UNI-DB` database schema.

Mapping between relationships and pairs of attributes

We define a mapping between relationships from $\Re$ and pairs of attributes from U^2 (expect key attributes) based on the mapping $\Psi_{\zeta\mathcal{D}}$ between attribute values and concepts in the ontology.

Definition 4.9 (Relationship-Attribute Pair mapping) *A mapping between the relationship set $\Re$ and a set of attribute pairs $(A_i, B_j) \in U^2$ is a relation $\Psi_{\Re\mathcal{P}} \subseteq \zeta \times U^2$ such that $\Psi_{\Re\mathcal{P}} = \{[\beta_i, (A_i, B_j)]\}$, where $\beta_i \in \Re$ and $(A_i, B_i) \in U^2$ are arguments in relationship correspondences.*

Given two attributes A and B of the same relation schema, the mapping of (A, B) to a particular relationship β_0 means that there is a data correspondence for each value of A and B so that their relevant attribute concepts are related through β_0. A mapping element can be derived from an expression of a relationship correspondence of type 2.b as in Figure 4.6. More specifically, based on Concept-Attribute and Concept-Value mappings the Relationship-Attribute Pair mapping satisfies the following condition.

Let $\beta_0 \in \Re$, $C_1, C_2 \in \zeta$, A, $B \in U$, and D_1, D_2 their domains, respectively.

Pair Constraints

$$H_5\textbf{:}\ \forall v \in D_1,\ w \in D_2$$
$$\Psi_{\Re\mathcal{P}}(\beta_0, (A, B)) \implies \exists\, c_v, c_w \in \zeta\ \ \Psi^*_{CA}(C_1, A) \wedge \Psi^*_{CA}(C_2, B) \wedge \beta_0(c_v, c_w) \wedge \Psi_{\zeta\mathcal{D}}(c_v, v)\ \wedge\ \Psi_{\zeta\mathcal{D}}(c_w, w)$$

Mapping between relationships and relations

A mapping between relationships β_i from $\Re$ and database relations R_i from Σ can be defined as a set of mapping elements (β_i, $R_1, \ldots R_k$). For each mapping element, a relationship can be mapped to one or more relation names. The mapping is then of type one-to-many, i.e., one relationship can be mapped to one or many relations. In this thesis we focus on one-to-one mappings.

Definition 4.10 (Relationship-relation mapping) *A mapping between the relationships from $\Re$ and relations from Σ is a relation $\Psi_{\Re\Sigma} \subseteq \Re \times \Sigma$ such that $\Psi_{\Re\Sigma} = \{(\beta_i, R_i)\}$, where $\beta_i \in \Re$ and $R_i \in \Sigma$ are arguments in relationship correspondences.*

First, if a mapping element associates one relationship with one relation, then this means that each instance of the relation extension represents a relationship between two entities of two different concepts. Thus, this mapping refers to a relationship correspondence of type 1 as in Figure 4.6. An example of such mapping element is ($Teach$, `Lecturer`).

However, if a mapping element associates one relationship with multiple relations, then this means that only a subset of the corresponding relation extensions are relevant to that relationship. This mapping refers to a relationship correspondence of type 3 as in Figure 4.6. An example of such mapping element is ($TeachCS$, { `Course`, `CS-Courses`}).

We assume that each decomposition of a given concept C in the ontology must reflect exactly the same decomposition for its associated value in the database instances. For instance, each computer science ('CS.') department modelled in the `UNI-DB` database should have two devisions 'Databases' and 'Networks'. These assumptions are formally expressed as follows.

Given a relation $R_1(A_1, \ldots, A_n)$, $R_2(B_1, B_2)$, $R_3(E_1, \ldots, E_s)$ and $R_4 = (F_1, \ldots, F_t)$ where A_1, (B_1, B_2), E_1, F_1 are primary keys, and B_1 and F_2, and B_2 are foreign keys referencing R_1, R_2, respectively. Let be $D_i = dom(A_i)$, $D_k = dom(F_k)$, and $D_j = dom(E_j)$ where $i = 2 \ldots n$, $j = 2 \ldots s$ and $k = 3 \ldots t$. The proposed axioms address two cases: (1) the database describes (N-M)relationship between entities and their parts (axiom H_1) and (2) the database describes (1-M)relationship between entities and their parts (axiom H_2).

Part Constraints

H_1: $\forall\, c\, p \in \zeta,\ a \in D_i,\ (x_1, \ldots, x_n, z_1, z_2, y_1, \ldots, y_s) \in R_1 \times R_2 \times R_3 :$
$\Psi_{\zeta\mathcal{D}}(p, a) \wedge R_1(x_1, \ldots, a, \ldots, x_n) \wedge Partof(p, c) \wedge \Psi_{\Re\Sigma}(PartOf, R_3) \implies$
$\exists h \in \zeta,\ b \in D_j \quad \Psi_{\zeta\mathcal{D}}(h, b) \wedge Isa(c, h) \wedge R_3(y_1, \ldots, b, \ldots, y_s) \wedge$
$R_2(x_1, y_1)$

H_2: $\forall\, c\, p \in \zeta,\ a \in D_i,\ (x_1, \ldots, x_n, y_1, \ldots, y_s) \in R_1 \times R_3 :$
$\Psi_{\zeta\mathcal{D}}(p, a) \wedge R_1(x_1, \ldots, a, \ldots, x_n) \wedge Partof(p, c) \wedge \Psi_{\Re\mathcal{F}}(PartOf, F_2) \implies$
$\exists h \in \zeta,\ b \in D_k\ \Psi_{\zeta\mathcal{D}}(h, b) \wedge Isa(c, h) \wedge R_4(y_1, x_1, \ldots, b, \ldots, y_s)$

H_3: $\forall\, c\, p\, p' \in \zeta,\ a \in D_i,\ (x_1, \ldots, x_n)\, (x'_1, \ldots, x'_n) \in R_1 :$
$\Psi_{\zeta\mathcal{D}}(p, a) \wedge R_1(x_1, \ldots, a, \ldots, x_n) \wedge Partof(p, c) \wedge Partof(p', c) \wedge \implies$
$\exists a' \in D_i\ \Psi_{\zeta\mathcal{D}}(p', a') \wedge R_1(x'_1, \ldots, a', \ldots, x'_n)$

The axioms H_1 and H_2 ensures that if a concept `p` is a part of a concept `c` and `p` is mapped to a value in the database, then there exist a value which is mapped to c or to an ascendant of `c`. The axiom H_3 ensures that if two concepts `p` and `p'` are parts of a same concept `c` and one of them is mapped to a value in the database, then the other must also be mapped to another value.

4.4 Mappings Discovery

For large applications, find a mapping form a database schema to another schema or from an ontology to another ontology becomes very difficult with manual approaches. Therefore, (semi)automatic techniques are required. For the database and AI communities this problem is often referred to as a *matching* problem.

A recent review of matching techniques can be found in [RB01, Noy04, PS05]. Most of the proposed techniques are not generic. They consider specific kinds of

data models such as relational schema, XML-schema, files, or taxonomies. Moreover, they are tailored to a specific application such as data integration and ontology merging [MBR01]. Unfortunately, these techniques often provide different results for the same matching task because they rely on different heuristics and metrics for measuring the matching accuracy. Nevertheless, to deal with this problem we can adapt and extend techniques of schema matching such as string-based similarity computations, graph-based representions of schemas, and use of thesauris to compare the meaning of words [MBR01, MGMR02, BSZ04]. We characterize the matching we intend to perform by the following features:

- It is *bi-directional*: We attempt to find elements of the database that can be mapped to those in the ontology and vice versa.
- It is one-to-one: It is very hard to automatically generate many-to-many mappings [RB01]. In this thesis we deal only with matching one element of the database with another one in the ontology.
- It may not be *total*: The proposed matching algorithms are *approximate*, i.e., they do not guarantee to provide all possible mappings. So, we may not find the mappings of all database elements in the results.

Formally the matching problem can be defined as follows.
Given a set of database elements E_{DB}, a set of ontology elements E_O, and a similarity metric, compute for each $(e, e') \in E_{DB} \times E_O$ mapping elements $< e, e', h >$ such that h represents the highest similarity between e and e'.

Obviously, the similarity values depend on the kind of the similarity measure we use and how it is computed. We consider two kinds of similarity metric (linguistic and semantic) as we shall see later.

4.4.1 Schema Model

To reason in a formal way on semantics of relational schemas, we associate a conceptional model with the database schema that can capture its schema properties. The basic idea is to express primary semantics that are embedded within schema elements. By adopting this model we can uniformly analyze the common semantics expressed in both the relational schema and the ontology. In addition, we represent this model as a labelled graph in order to allow the algorithm to find matchings based on similar graph-based layouts. Being inspired by the work proposed in the field of reverse engineering for transforming a relational database to an EER-model [CBS94], we build our schema graph based on the type of relation schemas, the type of their attributes, and their key characterizations. The key characterization of an attribute denotes whether it is a primary key, a foreign key or not a key. We refer to a primary key of a relation R as PK(R), to a foreign key as FK(R), and to a not key as NK(R). We classify database relations based upon the properties of their keys into three classes: *strong*, *weak*, *specialized*, *combined*, and *irregular* relations.

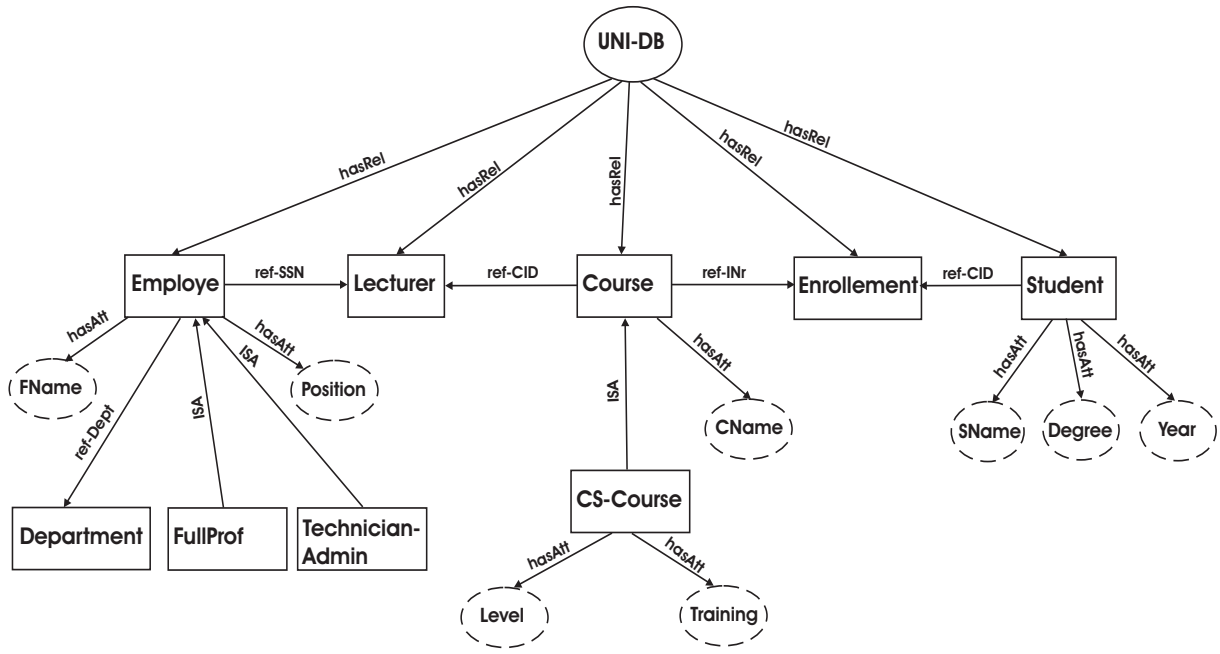


Figure 4.7: Dependency Graph of `UNI-DB` Schema

1. **A strong relation** is a relation whose primary key does not depend on primary keys of other relations, i.e., does not contain a foreign key. The relations `Student`, `Department`, and `Employee` are examples of strong relations.

2. **A weak relation** is a relation whose primary key depends on primary keys of other relations, i.e., a part of the primary key contains a foreign key.

3. **A specialized relation** is a weak relation whose primary key and foreign key are the same. For example, the relation `CS-Courses` is a specialized relation because its primary key `CS` references the primary key of the relation `Course`.

4. **A combined relation** is a relation which includes only a primary key that is formed by a concatenation of primary keys of other relations (strong or irregular). Examples are the relations `Lecturer` and `Enrollment`.

5. **An irregular relation** is a relation of other types, i.e., it is not of type strong, weak, specialized, or combined. An example of such relation is a relation whose primary key is a concatenation of two foreign keys and include additional attributes.

We note that strong relations may describe 1-to-N relationships between world entities they represent whereas combined and irregular relations may represent N-to-M relationships. For the rest of the thesis we refer to strong, weak, specialized, combined, and irregular relations as $R_{St} \in S_{St}$, $R_{Wk} \in S_{Wk}$, $R_{Sp} \in S_{Sp}$, $R_{Co} \in S_{Co}$, and $R_{Ir} \in S_{Ir}$, respectively.

We represent the model as a directed graph where nodes are labelled by the names of schema elements. We call it a *dependency graph* (DG). Figure 4.7 shows the dependency graph of the `UNI-DB` database schema. For the sake of clarity we omit some nodes and edges from this graph. Schema elements are relation and attribute names. A starting node is labelled by the database name. In Figure 4.7 the circle shape, oval shapes, and rectangles represent database name, attribute names, and relation names, respectively. Nodes are inter-connected by four kinds of edges. There are edges of type `hasRel` which relate the starting node to those of strong relations. There are edges of type `hasAtt` edges which relate a relation node with nodes of their attributes. Strong relation nodes are related to irregular relation nodes and composite nodes by edges labelled by a foreign key name prefixed by the string `ref`. For this purpose, we use key information to infer dependencies between relations. If a primary key of a relation, say ID, appears as a foreign key in another relation, then a `refID` edge is created between their corresponding nodes. Similarly, specialized relation nodes are related to each other and to strong nodes by edges of type `ISA`. Furthermore, inherited attributes of specialized relations will be pushed up to the strong relation in the `ISA`-taxonomy.

4.4.2 Matching Process

To match a relational database with an ontology we propose a matching process which includes a sequence of syntactic and semantic matching operations [9]. Our goal is to find appropriate mapping elements which could fulfill the requirements described in the last section. The matching process is *non-iterative*, i.e., it identifies matching elements in one pass. It consists of five steps:

1. Transform the relational schema into a dependency graph in order to capture the inter-dependencies between relations.

2. Perform similarity computations between all distinct pair of element labels from the database and the ontology. This step consists of three phases: the first phase carries out linguistic-based matching using syntax methods in order to check out whether the elements have similar labels. The second phase carries out structured-based matching using some heuristics in order to find similar nodes and edges. This phase can then support the linguistic-based one for determining another appropriate matchings. The third phase is a complement of the previous ones. It allows to improve the matching results previously found and to discover other matchings for specialized relations.

3. The result of the matching is not a mapping on its own, but rather an approximation of the similarity between elements from the database and the ontology. Nevertheless, the computed pairs have high chance of being selected for the final result. One of the ways to select the correct pairs is to display the element pairs with their similarity scores and ranks, and leaving the choice of the appropriate pairs up to DBAs (DataBase Administrators).

[9]The corresponding matching operator is often called a matcher

4. Make refinement and validation of the mappings. Here, the DBAs have to check whether all database elements are mapped into the corresponding ontology elements, i.e., whether the matching is complete.

5. Find appropriate mapping elements based on the matching results. For this purpose some existing methods could be used, e.g., the value correspondence method proposed in Clio [HMH01].

4.4.3 Linguistic Matching

The linguistic matching takes advantage of the structure of a string (i.e., a linear sequence of characters or words or phrases). The matching is based on the assumption that labels used for ontology and database elements which have similar syntactic features probably denote the same things. That is, the matcher uses a syntactic measure to compute the closeness between the labels of two elements e_i and e'_j, linguistically or structurally. In this case, the result of a matching has the form $< e_i, e'_j, m >$, where m is a coefficient, typically, a number in [0,1]. The higher m is, the more similar e_i and e'_j are. Similarity computations are performed in two stages as follows.

Linguistic analysis. We assume that labels are strings that have meaning in the same natural language. Before comparing the strings, we have to perform some primarily normalization procedures that help improving the results of the comparison:

- *Tokenization*: Breaks each string into tokens, eg., $CS{-}Courses \rightarrow \{CS, Courses\}$.
- *Part of speech tagging*: Replaces each token by its lexical category (a tag) among those proposed in the tokenization [Bri94]. For instance, the word "match" could be a verb or a noun. Thus, the terms "matching", "matched", "matches" would be reduced to a single tag "match" .
- *Expansion*: Expands abbreviations and acronyms of each token, eg., $\{CS \rightarrow ComputerScience\}$.
- *Case normalization*: Converts each character in tokens into its lower case (or upper case) counterpart.
- *Suppression*: Consists of eliminating punctuation, digits (e.g., $\{courses2 \rightarrow courses\}$) and stop words (eg., "a", "the", "to", "of").
- *Stemming*: Conbines the variant forms of a word into a common representation called the *stem*. Example of stemming are translating the plural forms into their singular (e.g., $\{courses \rightarrow course\}$), transforming feminine forms to their masculine, and removing accents [CX95, OH01].

The output of this analysis can be stored in a look-up table for further processing steps.

Input: $DBset$, $ONset$ {Two sets of database and ontology labels}
Output: SimilarityTable[]

1: $TokenTable \leftarrow PrepToken(DBset)$ {preprocessing labels to build tokens}
2: $TokenTable \leftarrow PrepToken(ONset)$
3: **for all** $l_i \in DBset$ **do**
4: $Tlist_i \leftarrow TokenTable.Lookup(l_i)$
5: **for all** $l_j \in ONset$ **do**
6: $Tlist_j \leftarrow TokenTable.Lookup(l_j)$
7: $score \leftarrow ComputeSimilarity(Tlist_i, Tlist_j)$
8: $SimilarityTable \leftarrow (l_i, l_j, score)$
9: **end for**
10: **end for**
11: Sort($SimilarityTable$) {sort table entries by score }
12: **return** $SimilarityTable$
13: Function $ComputeSimilarity(T_1, T_2)$
14: **for all** $t_i \in T_1$ **do**
15: **for all** $t_j \in T_2$ **do**
16: $score \leftarrow StringMatch(t_i, t_j)$
17: **if** $maxScore < score$ **then**
18: $maxScore \leftarrow score$
19: **end if**
20: **end for**
21: $tScore \leftarrow tScore + maxScore$
22: **end for**
23: $tScore \leftarrow \frac{tScore}{max(\|T_1\|,\|T_2\|)}$
24: **return** $tScore$

Algorithm 1: The Linguistic-Matching Algorithm

Similarity comparison. To compare tokens we choose a string matching algorithm based on the *substring similarity* measure [Mel95]. The intuition behind the substring similarity is that strings would be very similar when they share a common substring. We consider the longest common substring (LCS) of characters (do not necessarily need to be contiguous). The similarity between two strings is then estimated by dividing the length of the LCS by the sum of the lengths of the strings in order to obtain a normalized score. Following Efraty et al. [EL04] we define the simiarity score as follows.

Definition 4.11 (Substring similarity) *Let S be a set of strings. The substring similarity is a function $\sigma : S \times S \longrightarrow [0, 1]$ such that* [10]*:*

$$\forall\ x,\ y \in S\ ,\ let\ l\ be\ the\ LCS\ of\ x\ and\ y,\ \sigma(x, y) = \frac{2|l|}{|x|+|y|}$$

Given two element labels l_1 and l_2 and their corresponding token sets T_1 and T_2. The matcher calculates the similarity score of all pairs of tokens from T_1 and T_2 based on the

[10] $|s|$ denotes the length of a string s

similarity measure above. Then, it computes the similarity of labels using the maximal similarity score of tokens (based on Monge-Elkan formula [ME96]) as follows [11].

$$\delta(l_1, l_2) = \frac{\sum_{t_i \in T_1}[max_{t_j \in T_2}\sigma_i(t_i, t_j)]}{max(\|T_1\|, \|T_2\|)} \tag{4.1}$$

The linguistic matching is represented formally as ALGORITHM 1. The algorithm takes as input two sets of ontology labels and database labels and returns as output a similarity table where its entries are sorted by score. Using this table the DBAs will choose only the entries whose score is above a given threshold. The function $PrepToken(L)$ extracts tokens from label set L and filters them as described earlier. Function $ComputeSimilarity(T_1, T_2)$ computes similarities between tokens T_1 and T_2 according to equation 4.11. The function $StringMatch(t_i, t_j)$ return the similarity score between the strings (t_i and t_j) as defined above. The algorithm runs in $O(n^2)$ time, where n is the maximum number of database or ontology labels.

4.4.4 Structural Matching

Structural matching computes similarity between two elements based on their node positions within their corresponding graphs. The intuition behind these techniques is that if two nodes from different graphs are similar, then their neighborhood nodes are also similar. There are many heuristic rules that can capture this intuition [ES04]. An example of such rules is "two nodes are structurally similar if their immediate children nodes are highly similar". In this thesis we develop heuristic rules which are derived from the semantics represented in the graphs. These rules are described in detail as follows.

- **R1.** If two strong relation nodes R_{St} and R'_{St} matches two ontology nodes C_1 and C_2 respectively, then the edge connecting R_{St} and R'_{St} matches an edge in the path between C_1 and C_2.
- **R2.** If two strong relation nodes R_{St} and R'_{St} match two ontology nodes C_1 and C_2 respectively, then the combined relation node R'_{Co} connected to both R_{St} and R'_{St} match an edge in the path between C_1 and C_2.
- **R3.** If two strong relation nodes R_{St} and R'_{St}, which are related to an irregular relation node Z_{Ir}, match two ontology nodes C_1 and C_2 respectively, then the node Z_{Ir} match a children node common to C_1 and C_2.
- **R4.** Given two ontology outgoing edges β_1 and β_2 whose end nodes are related through an ISA edge, if β_1 match a composite relation node R_{Co} and the end node of β_2 matches a specialized relation node R'_{Sp} then the edge β_2 matches both R_{Co} and R'_{Sp}.
- **R5.** If two specialized relation nodes R_{Sp} and R'_{Sp} having a common adjacent (strong relation) node R_{St} matches two ontology nodes C_1 and C_2 respectively, then the R_{St} node matches an ascendent node common to C_1 and C_2.

[11] $||X||$ denotes the cardinality of a token set X

Example 4.1 (Matching EMO and DG of UNI-DB) An example for the rule **R1** is that, if we match the graph associated with the ontology EMO and the database schema of UNI-DB we can deduce that the relation nodes Employee and Department could be matched with the concept nodes EMPLOYEE and DEPARTMENT, respectively. Therefore, we can infer that the edge refDept could be matched with the edge WorkIn implying the same meaning.

It is important to note that if the path between EMPLOYEE and DEPARTMENT contains ISA edges than these edges will be ignored because the semantic of ISA is dominated by other relationship semantics. An example of such path may be " EMPLOYEE—> WorkIn—> DIVISION — ISA–> DEPARTMENT". Similarly, the relation node Course could be matched with the ontology node COURSE. By examining the path between the nodes EMPLOYEE and COURSE we can deduce that the relation node Lecturer could be matched to the edge Teach (w.r.t. **R2**).

On the other hand, if the relation node CS-Courses is matched with the ontology node CS.COURSE, then we can infer that the edge TeachCS could be matched with the relation nodes Lecturer and CS-Courses (w.r.t. **R4**). Semantically, the instances represented by TeachCS in EmO are modelled as the result of a join operation on the relations Lecturer and CS-Courses. Furthermore, the relation nodes FullProf and Technician-Admin could be matched with the concept nodes PROFESSOR and NON ACADEMIC, respectively. Therefore, according to rule **R5** the related node Employee could be matched with the common concept node EMPLOYEE.

The structural matching has to apply the above rules in order to find matching elements related to Concept-Relation $\Psi_{\zeta\Sigma}$, Relationship-ForeignKey $\Psi_{\Re\mathcal{F}}$, and Relationship-relation $\Psi_{\Re\Sigma}$ mappings. □

4.4.5 Semantic Matching

Semantic matching is concerned with the semantic correspondence between two given labels e_i and e'_j. That is, the matcher computes a logical relation between the entities represented by e_i and e'_j. In this case, the result of a matching has the form $< e_i, e'_j, m >$ where m is a semantic relationship of the form ($\equiv$), more general/specific ($\sqsubseteq, \sqsupseteq$), or mismatch ($\perp$).

We use a technique used in information retrieval for integrating documents about product catalogues [12] [BSZ04]. Our goal is to discover logical relations between elements by means of making explicit their meaning. This can be achieved by the assistance of some dictionaries or thesauri like WordNet [Fel98] and Nexus [Jan01], which store the necessary linguistic and domain knowledge about lexical terms. For instance, WordNet is an electronic lexical collection of English words, where various meanings of words (*senses*) are grouped by sets of synonyms (*synsets*). For example, the word "administrator" has three senses *administrator*♯1 (someone who administers a business), *administrator*♯2 (someone who manages a government agency or department), and *administrator*♯3 (the party appointed by a probate court to distribute the estate of someone who dies without a will or without naming an executor). Synsets are in turn

[12] Products are classified under categories

	ACADEMIC *academic♯*	**PROFESSOR** *professor♯*	**TECHNICIAN** *technician♯*	**EMPLOYEE** *employee♯*
Full-Prof *full professor♯*	$\sqsubseteq$	$\equiv$	$\bot$	$\sqsubseteq$
Technician-Admin { *technician♯*, *administrator♯* }	$\bot$	$\bot$	$\sqsupseteq$	$\sqsubseteq$
Employee *employee♯*	$\sqsupseteq$	$\sqsupseteq$	$\sqsupseteq$	$\equiv$

Table 4.1: Computed Matrix for matching elements of EmO and UNI-DB

classified in a taxonomy. A synset could be related to more general a synset (through a *hypernym* relationship) and to more specific synsets (through a *hyponym* relationship). In addition, other relationships like antonymy and meronymy [13] are also available.

The semantic computation is carried out in two phases: *semantic interpretation* and *semantic comparison*. We describe them in details as follows.

Semantic interpretation. In this stage, for each element label we provide an interpretation of their tokens, i.e., each token is associated with a set of senses. Tokens that are prepositions, or commas, or conjunctions allow to build a unified sense for the labels. For instance, the label $L =$ "*books and papers*" can be parsed into three tokens $\{book, and, paper\}$. Therefore, L will be semantically interpreted by senses of both *book* and *paper*, i.e., $sense(L) = sense(book) \cup sense(paper)$. For elements represented in ISA-Taxonomies, senses are further filtered in two ways. First, we have to remove senses that contradict each others. This is done by considering senses of their ancestors and descendants as follows. For a given element e whose label is L having an associated sense set $sense(L) = \{s\sharp 1, \ldots, s\sharp n\}$, a sense $s\sharp i$ can be deleted from the set if one of the following situations occur:

- There is an ancestor e' of e having a sense $s'\sharp j$ such that $s'\sharp j$ is less general than $s\sharp i$.
- There is a descendant e' of e having a sense $s'\sharp j$ that $s'\sharp j$ is more general than $s\sharp i$.
- There is a parent e' of e such that $s\sharp i$ is opposite to any sense $s'\sharp j$ of e'. That is, $s\sharp i$ and $s'\sharp j$ are different hyponyms of the same synset or they are antonyms.

Second, we have to filter the resulting senses. Sense filtering combines the structural information of the node label with its interpretation. That is, if two nodes are siblings then we can eliminate senses that involve homonym/hypernym relation. Formally, given two sense sets $S = \{s\sharp 1, \ldots, s\sharp n\}$ and $S' = \{s'\sharp 1, \ldots, s'\sharp n\}$ of two sibling nodes. If there exists $s'\sharp j \sqsubseteq s\sharp i$, then $s\sharp i$ is replaced by $s\sharp i \backslash s'\sharp j$.

[13] Called also part-of or holonymy

Input: G_1, G_2 node graphs {two graphs of nodes of type *node*}
Output: SimilarityMatrix[]

$node : struct\{id : int, Nlabel : string, Ntoken : List, Ntype : string\}$
$Initialize(SimilarityMatrix[n_1, n_2])$ {fill the matrix with empty strings}
$PrepGraph(G_1)$, $PrepGraph(G_2)$ {preprocessing node labels to build tokens}
for $i = 1$ To n_1 **do**
  for $j = 1$ To n_2 **do**
    $SimilarityMatrix[n_i, n_j] \leftarrow ComputeSimilarity(G(i).Ntoken, G(j).Ntoken)$
  end for
end for
return $SimilarityMatrix[n_1, n_2]$
Function $ComputeSimilarity(T_1, T_2)$
$S_1 = GetSenses(T_1)$, $S_1 = GetSenses(T_2)$
for all $s_i \in S_1$ **do**
  for all $s_j \in S_2$ **do**
    if $[s_i = s_j]$ **then**
      return "$\equiv$"
    end if
    if $[isHypo(s_i, s_j) \,|||\, isMero(s_i, s_j)]$ **then**
      return "$\sqsubseteq$"
    end if
    if $[isHyper(s_j, s_i) \,|||\, isMero(s_j, s_i)]$ **then**
      return "$\sqsupseteq$"
    end if
    if $isOpposite(s_i, s_j)$ **then**
      return "$\perp$"
    end if
  end for
end for

Algorithm 2: The Semantic-Matching Algorithm

Example 4.2 For instance, we suppose that employees are classified into two relations `Academic` and `Professor`. If we assume that the $sense(Academic) = \{academic\sharp1 (professor, assistant)\}$ and $sense(Professor) = \{professor\sharp1(professor)\}$, then `Academic` will be reinterpreted as `Academic` with the exception of `professor`. That is, the instances in `Academic` relation are in fact about assistants. □

It is worthy noting that this filtering is valid only under assumption that instances of relations are disjoint.

Semantic comparison We represent a semantic matching using a matching matrix M where columns and row heads designate filtered senses of element labels. Given a database label s and a ontology label t, and let $sense(s) = \{s\sharp1, \ldots, s\sharp n\}$ and

$sense(t) = \{t\sharp 1, \ldots, t\sharp n\}$ be their senses, respectively. Following Bouquet et al. [BSZ03] we can determine the matching value of $M(s,t)$ by applying the following operations:

- If $s\sharp i = t\sharp j$ then set $M(s,t) =$ " $\equiv$ ".
- If $s\sharp i$ is either a hyponym or meronym of $t\sharp j$ then set $M(s,t) =$ " $\sqsubseteq$ ".
- If $t\sharp i$ is either a hypernym or meronym of $s\sharp j$ then set $M(s,t) =$ " $\sqsupseteq$ ".
- If $s\sharp i$ and $t\sharp j$ have opposite meanings then set $M(s,t) =$ "$\perp$".

where $1 \leq i, j \leq n$ and $s\sharp i$, $t\sharp j$ are any sense in $sense(s)$ and $sense(t)$, respectively.

Table 4.1 represents the output of semantic matching for some elements of the ontology `EmO` and the database schema `UNI-DB`. Each intersection between a row and a column contains a semantic relation.

The semantic matching is represented formally as ALGORITHM 2. The algorithm takes as input two graphs (ontology graph and dependency graph) and returns as output a similarity matrix where matrix values are logical relations. The function $PrepGraph(G)$ proceeds nodes of graph G to build tokens.
The function $ComputeSimilarity(T_1, T_2)$ computes logical relations between two token sets T_1 and T_2. First, it applies function $Getsenses(T_i)$ which constructs filtered senses of each set T_i, $(i = 1, 2)$. Then, it tries to find relationships between any tokens from distinct labels using WordNet domain knowledge. These knowledge are expressed by $isHypo()$, $isHyper()$, $isMero()$, and $isOpposite()$ operations as described above.

The algorithm runs in $O(n^2)$ time, where n is the maximum number of graph nodes. The results of the matching algorithm are used for two purposes: First, to improve the output of linguistic matching. This is done by increasing the similarity score of element labels that are synonyms (set to 1). Second, to find matching between two `ISA`-taxonomies from the ontology and schema graphs. The claim is that syntactic matching do not perform very well alone because `ISA` related elements have very similar meaning and their structural locations in the graph impact the meaning. On the other hand, semantic matching often need identifying some nodes to perform well. These nodes, which are located in `ISA`-taxonomy of different graphs, are declared to be equivalent. They are called *anchors*. Identifying these nodes can be carried out automatically based on the output of linguistic matching or manually based on the DBAs input.

4.5 Related Work

4.5.1 Matching Approaches and Systems

This section shortly presents some matching approaches related to our work and illustrates why we need to develop a new approach for matching ontologies and databases. The literature references various approaches and systems which have been developed to semi-automatically solve the matching problem, e.g., Cupid [MBR01], Artemis [CAdV01], SF [MGMR02], LSD [DDH01], O2R [BCP04], and D2R [Biz03].

While most of them address schema-to-schema problem, a few approaches address ontologies-to-databases problem (e.g., O2R and D2R). In [RB01] Rahm et al. provide a classification of the matching techniques based on the following common features:

- *Individual* vs. *combining*: An Individual matcher uses one criterion to compute mappings. A combining matcher can be either a *hybrid matcher* or a *composite matcher*. A hybrid matcher uses multiple criteria whereas a composite matcher combines results obtained by using several matching algorithms.
- *Instance-based* vs. *schema-based*: While instance based matchers consider only data instances, schema based matchers consider only schema-level information.
- *Element-level* vs. *structure-level*: Element-level matchers compute mappings by considering elements in isolation (e.g., single attributes), ignoring their relations with other elements. Structure-level matchers compute mappings by considering how elements appear together in a structure.
- *Language* vs. *constraint-based*: Language matchers rely on linguistic techniques, e.g., comparing names of elements. Constraint-based matchers exploit the constraints being applied to defining schema elements, e.g., keys.
- *Matching cardinality*: The matching may result in relationships between one or more elements of one schema and one or more elements of another schema. There are four kinds of relationships: 1:1, 1:n, and n:m.
- *Additional resources*: A matcher might exploit auxiliary resources expressing the relationships between element names, e.g., thesauri.

Cupid. The Cupid algorithm [MBR01] has been developed for discovering mappings between relational or XML schema elements. It depends, among others, on the name and data type similarities of the attributes. Similarity coefficients are computed with the assistance of a domain specific thesauri. Identical elements have the highest similarity value. Cupid represents a hybrid matching approach by combining successively a linguistic and structural schema matching techniques. The linguistic matching is used to compute linguistic similarity coefficients between schema element names based on morphological normalization, categorization, string-based techniques (common prefix and suffix) and a thesaurus look-up. The structural matching transforms the original schema into a tree and then performs a bottom-up matching resulting in structural similarity between pairs of element. The basic idea of the structural matching is to rely on matching the leaf elements instead of matching the immediate descendants or intermediate substructures when we compute similarity between non-leaf elements. Afterwards, Cupid determines similarities by using weighted coefficients and produces final mappings by choosing pairs of schema elements having coefficients higher than a given threshold.

Artemis. Artemis [CAdV01] has been designed as a module of MOMIS [BBC+00] mediator system for integrating schemas. It represents a hybrid approach and operates on object oriented models. Artemis performs affinity-based analysis and hierarchical clustering of schema elements. It calculates the name, structural, and global affinity coefficients exploiting a common thesaurus. The thesaurus, which is created based on WordNet, presents a set of terminological and extensional relationships (e.g., synonym, hypernym). These relationships capture intra- and inter-schema knowledge about classes and attributes of the input schemas. A hierarchical clustering technique exploiting global affinity coefficients categorizes classes into groups at different levels of affinity. For each cluster it creates a set of global attributes and hence creates a global class.

SF. The Similarity Flooding (SF) [MGMR02] approach has been developed to match diverse schemas including relational schema, UML [Gro05], and RDF [BG06]. It presents a hybrid matching algorithm which uses a method called similarity propagation. The algorithm performs name matching, which suggests an initial element-level mapping which will be complemented by the structural matching. To this end, SF first converts schemas into labeled graphs based on the OIM specification [Cov99]. These graphs are used in an iterative fixpoint computation to find 1-to-1 correspondences of local cardinality and m-to-n correspondences of global cardinality between related nodes in the graphs. Unlike other approaches, SF does not exploit terminological relationships in an external thesaurus, but only relies on string similarity between element names (common prefix and suffix). Second, the algorithm compares the vertices labels based on their string similarity in order to obtain an initial mapping. This mapping will be further refined within the fix-point calculation. SF is based on the idea of computing similarity coefficients between similar nodes using coefficients of their adjacent neighborhood nodes. The similarity computation is done for a finite number of iterations until the similarity measure reaches the fix-point. In the last step, various filters are applied to select the most relevant mappings.

LSD. LSD (Learning Source Description) [DDH01] has been developed to integrate XML schemas. LSD represents a composite approach to combine different matching algorithms. The algorithms use machine learning techniques to find 1-to-1 element-level mappings. Additionally, the algorithms use various instance-level matchers during a preprocessing phase (learning phase). The matchers are based on learning classifiers for classes from instances in order to evaluate the probability distributions of instances. They use different criteria to evaluate the reason to belong to a class. After the learning phase, LSD attempts to discover various characteristics of patterns related to elements of the target schema. The resulting patterns and rules are combined by a meta-learner which weights the predictions from a matcher according to its accuracy during the training. Although the approach is primarily instance-oriented, it can also exploit schema information. An instance-level matcher takes a self-describing pattern and matches it using only tags in the schema rather than the instance values.

		Cupid	Artemis	LSD	SF
Schema types		XML, relational	ER, OO, relational	XML	UML, OEM,RDF relational
Matcher type		hybrid	hybrid	composite	hybrid
Matching cardinality		1:1 , n:1	1:1	1:1	1:1 , n:1
Schema level	**Name based**	name equality, Abreviations, synonyms	name equality, synonyms	name equality, synonyms	name equality, synonyms,
	Constraint based	data type, domain compatibility, ref. constraints	domain compatibility, Keys, aggregation	—	typing and cardinality constraints
	Structure based	subtrees weighted by leaves	neighborhood matching	XML classifier for matching non-leaves	neighborhood matching
Instance level	**Text based**	—	—	Whirl, Bayesian learners	—
	Constraint based	—	—	List of valid domain values	—
Semantic based		—	—	—	—

Figure 4.8: Features of matching systems

R2O. R2O [BCP04] is a language developed to describe mappings between relational schemas and ontologies implemented in RDF(S) [BG06] or OWL [MH06]. The important idea behind this approach is that mappings between the schema elements and the corresponding concepts of the ontology will be defined declaratively in a mapping document. This mapping document will be the input of a processor charged of carrying out an effective populating of an ontology with instances extracted from the database content. The fact of defining these mappings declaratively will make the solution domain independent and reusable. Using R2O, a mapping between a database schema and an ontology is defined as a set of mapping expressions like $OC_i = Transformation(DBE_j, DBE_k)$, where OC_i is an ontology concept, and DBE_j and DBE_k are relations or attributes. Details on the language and the use cases can be found in [BCP04].

Comparison with our approach. We summarize the features of the above matching systems in the table shown in Figure 4.8. Like Cupid, Artemis, and SF we use a hybrid approach comprising linguistic, structural and semantic techniques. Like LSD and Artemis, the cardinality of matching is 1:1 as described in Section 4.3. The major difference between these approaches and our approach is that they intend to find mappings between two schemas. However, our goal is to both map a database schema and its content to a given ontology. Thus, we also have to perform an instance-level matching between database instances and ontology concepts. Furthermore, we propose a semantic matching between schema and ontology elements based on their meaning. Up to now, to our knowledge there is no approach that addresses directly the problem

of matching an ontology with a database. Even if we apply the existing technique of schema-to-schema or ontology-to-ontology matching to this problem, the match results would not be promising because of these main reasons: (i) ontologies and database schemas have different abstraction levels for representing data semantics (e.g., ontologies contain several hierarchical structures), (ii) in particular `ISA`-taxonomy is the main feature of an ontology structure, (iii) in general ontologies do not contain instances, (iv) and they also do not have keys or instance identifiers such that instance based matching will not be possible. We should also mention that it is difficult to prove that our approach is 'better' (or 'not better') than the existing approaches. Benchmarks and systematic methodology for evaluating different matching algorithms based on well-defined measurements do not exist [MGMR02]. In general, it is hard to compare the matching algorithms implemented in the different systems since most of them are not generic but tailored to a specific application domain and schema types. Moreover, it is difficult to find two systems evaluated on the same dataset. The evaluations are based on diverse quality metrics making it difficult to assess the overall effectiveness of the systems [MGMR02].

For representing the mappings, the R2O approach presented in [BCP04] relates closely to our work. However, the major difference between R2O and our approach is that we are given an existing ontology and want to map it to an existing database schema and instances, whereas R2O aims at constructing a new ontology. In addition, the language provided by R2O have limited expressiveness because it does not enable representing the mapping of database instances. Regardless this limitation, we benefit from the declarative mapping techniques used in this language to develop our declarative mapping language as described in Section 6.2.

4.5.2 Ontology-based Approaches and Systems

This section shortly describes some of the existing prototype systems related to our work. We consider the most important aspects used to achieve semantic interoperability. These aspects includes:

- Type of the system architecture: This aspect refers to the components of the system which are significantly influenced by the use of ontology (see Section 4.2) and the type of information sources supported (files, databases, etc.).
- Use of ontologies: This aspect refers to how ontologies are exploited to solve semantic problems, how they are represented, and how they are mapped on the underlying sources.
- Query resolution: This aspect refers to the set of query processing steps in which ontologies are involved to answer a query.

SIMS. SIMS (Search In Multiple Sources) [AHK01] is based on the *mediator/wrapper-based* architecture. This architecture consists of a mediator and a set of wrappers. A *mediator* provides users with an integrated access to a set of heterogenous sources. Each

source is accessed through a *wrapper*. A wrapper in SIMS is a program which translates a query expression written in SIMS's internal language into a form appropriate for the source. The wrapper also handles communication with the information source and collects the query results and returns them to the mediator. Currently, SIMS supports sources which are relational databases including Oracle, Ingress, and Informix. SIMS adopts a single ontology approach (with regard to the classification presented in Section 4.2). The SIMS mediator contains a domain schema which provides a general terminology for a particular application domain, called also a global ontology, as well as information about the structure and content of sources themselves. In SIMS, each information source is defined in terms of the domain schema. This approach is called *local-as-view* (LAV). The ontology is represented in LOOM language [14] and consists of a set of concepts. In SIMS, a concept is expressed as a class associated with attributes. A class can have a number of subclasses but SIMS does not allow multiple hierarchies.

In order to establish a connection between the domain model and the information sources SIMS uses one-to-one mappings: a mapping between one relation name of the source and one class name used in the domain model, and a mapping between one attribute name used in the source and another used in the domain model. Therefore, a change in the structure of the source implies a modification of the ontology. For instance, if a relation represents only a subset of instances of a class then a new subclass would be created in the ontology and the relation name would be linked to the name of that subclass.

SIMS is designed to allow users to formulate queries in terms of the domain schema without specific knowledge about the information sources and how they are related to the domain schema. To answer a query the system applies firstly four reformulation operations: *Select-Information-Source*, *Generlalize-Concept*, *Specialize-Concept*, and *Decompose-Relation*. The first operation is used to map directly domain schema concepts to source elements. The second operation uses information about a class and its superclass for the query reformulation when the information source does not provide information about that class. The third operation uses information about subclasses in the same way as the previous operation. The fourth operation replaces a relation invoked in a query with the elements of the source that specify its corresponding instances. Once the query is transformed the system divides it into sub-queries and send them to the appropriate wrappers [AHK01].

There is no formal theory for this system and the completeness features of the transformation methods have not been investigated in detail.

TAMBIS. TAMBIS (Transparent Access to Multiple Bioinformatics Information Sources) [PSB+99] is the first system developed to provide an unified to access biological heterogenous sources. Generally, data sources are not structured databases but rather Web-based sources (e.g., HTML-files, XML-files). Unlike SIMS, TAMBIS adopts *a global-as-view* (GAV) approach, where a global ontology is defined in terms of local views on the source schemas. The global ontology, called TaO, currently contains around 1800 biological concepts and relationships. It describes different features

[14] A member of the KLONE family of knowledge representation systems

of proteins (e.g. structure, classification), nucleic acids, and biological functions and processes.

Like SIMS, the TAMBIS architecture is based on the mediator/wrapper model. The main components of the system are: *a user interface*, *an ontology server*, *a Sources and Services Model* (SSM), *a query planner*, and *wrappers*. The user interface is entirely driven by the ontology where users browse the ontology and formulates their queries without being aware of relevant sources. Concepts in queries could be specialized or generalized using the subsumption hierarchy of the ontology. The ontology server is responsible to store and to represent the ontology using a description logic languages [15]. It also supports various reasoning services over the ontology concepts and their inter-relationships, and control the combinations of terms by inferring constraints encompassed in the ontology known as *sanctioning*. For example, the sanctioning mechanism of GRAIL allows to validate query expressions before retrieving data from underlying sources. It also allows both abstract and intensional answers to be returned. This is done by exploring legitime relationships between concepts and hence restricting users to certain shared terminologies such that the formulated queries are biologically meaningful [GSN+01]. The query planner uses the SSM to identify the sources used to answer the query and constructs a query plan to be efficiently evaluated. The SSM provides mappings between concepts and relationships in the ontology, and the services provided by the sources (wrapper services). These services have the form of function calls expressed in the Collection Programming Language (CPL) [BDH+95]. In SSM, the CPL functions are indexed by terms from the ontology and are used to evaluate the formulated queries. There are three main categories of functions described in SSM: *Iterators* which retrieve instances of concepts from a source, *Role evaluator* which evaluates instances participating in relationships and compute their values, and *fillers* which are used to discard instances that are not relevant to the query. Other kinds of mappings are manually stored in lookup tables and have the form of single valued mappings ($concept \times source \rightarrow string$) or the form of set valued mappings ($concept \times source \rightarrow strings$) [PSB+99].

Carnot and InfoSleuth. The Carnot research project [WCH+93] has been founded in 1990 in order to deal with the problem of logically unifying distributed information sources of enterprises. Like SIMS, Carnot is based on the mediator/wrapper architecture. The system supports object-oriented and relational databases. The mediator uses an ontology, called CYC [LG90], to capture semantics of the different heterogenous databases. The CYC ontology describes very general concepts that are common across many application domains. It is implemented in the CycL language [LG90]. Therefore, Carnot adopt the single ontology approach as described in Section 4.2. Carnot defines mappings between each individual source and the global schema using a set of *articulation axioms*. Articulation axioms define the relationships between the terms used in the source and the terms of the global schema. These axioms are build using CYC but after that the ontology is no longer used or needed. In contrast, the ontology of SIMS is part of the system and is used for each query submission. For each SQL query a graph

[15] The previous version uses GRAIL [RBG+97] but actual version uses ALCQI [CLN99]

is generated using the articulation axioms. This graph is expanded by the mediator including sources with relevant information.

The InfoSleuth project [BBB+97] is build on the Carnot technology to make database systems easily accessible via the internet-based World Wide Web. InfoSleuth employs a collection of agents for integrating data in a dynamically changing network of information sources. An agent is a software process that activates external services or encapsulates internal algorithms. Multiple agents interact with each other and cooperate to solve complex information tasks, e.g., routing, analysis, extraction, and integration. Agents in InfoSleuth are classified into three categories: *user agents*, *source agents*, and *core agents*. All agents use ontologies to communicate one with another. The ontologies describe the application domain including domain objects, events and activities. User agents assist users to formulate their queries over an ontology and to convert them into a form understandable by the agents themselves, and display the result of the queries in a form accessible for the users. Source agents are wrappers that translates queries and data stored in the sources from their local forms into the InfoSleuth form. These agents usually need mappings between the the ontologies and the local schemas. The core agents are responsible for gathering information needed to process the user queries, and synthesizing, filtering and abstracting that information into the level of abstraction that the user needs [NFK+00]. Regarding the classification of Section 4.2 InfoSleuth adopts the multiple ontology approach. The ontologies of InfoSleuth are specified in the OKBC [CFF+98] language standard and are stored in an OKBC server. The server is accessed by user agents to provide specifications to users for query formulation and is also accessible by source agents to perform mappings between the ontologies and the terms used in the local sources. Thus, a resource agent presents information of a source to the system in terms of one or more ontologies. However, there is no specific information about how mappings are defined in InfoSleuth.

OBSERVER. OBSERVER (Ontology Based System Enhanced with Relationships for Vocabulary hEterogeneity) [MKSI96] is a system that uses multiple ontologies approach to access data repositories [16]. Each repository contains several distributed data sources. The sources can be dynamic and can be built in different formats and languages. The architecture of the system is based on four modules: *wrappers*, *ontology servers*, and an *IRM* (Inter-ontology Relationship Manager). The task of the wrapper is to retrieve data from the underlying source and to hide its specific organization to the rest of the system. A repository contains also a set of wrappers, a set of ontologies, and an ontology server. The task of an ontology server is to provide information about the ontologies of its repository including mappings between an ontology and the structure of its underlying source. The IRM can be seen as a catalog of semantics of the system used to deal with the vocabulary problem.

In OBSERVER, ontologies can be seen as a set of metadata that represents intentional descriptions (i.e., structure and organization of source content) of the underlying sources. Ontologies are expressed in description logics [17]. Users formulate queries

[16] In OBSERVER data repositories are defined as a set of entity types and attributes

[17] Currently the prototype uses the language CLASSIC [BBM+92] but other languages are also supported

over one ontology (called *user ontology*) by choosing the appropriate terms from its definition. The system has the responsibility of managing the heterogeneity and distribution among data sources. This is done by translating query terms into terms in other ontologies using terms from the IRM and the ontology server. The system uses two types of mappings in query processing: one type links each term in an ontology with structures in a data repository and other type (called *inter-ontology* relationships) relates the terms used in different ontologies. The first type is used by the ontology server to translate a query into different sub-queries to the underlying data repository. The wrappers retrieve the data corresponding to each sub-query from the sources and the answer is returned to the user in a common format. If the user wants to obtain more relevant data, then the IRM can translate the query in terms of another ontology using the inter-ontology relationships. Obviously, this translation is not always exact and may imply a loss of information. There are six kinds of inter-ontology relationships [MKSI96]:

1. Synonym relationships: When two terms in different ontologies have the same meaning.

2. Hyponym relationships: When a term in an ontology has less general meaning than other term in another ontology.

3. Hypernym relationships: When a term in an ontology has more general meaning than other in another ontology.

4. Overlap relationships: When there exists an intersection between the real-world entities represented by two terms in different ontologies.

5. Disjoint relationships: When there exists no intersection between the real-world entities represented by two terms.

6. Covering relationships: When a set of entities represented by a term in an ontology is the same as an union of entities represented by other terms in other ontologies.

COIN. COIN (COntext INterchange) [DGH+96] is a system that provides access to structured data source such as databases, as well as semi-structured sources such as Web sites. The system is based on the mediator approach. It introduces a novel for the mediated data access in which the notion of *context* is used to detect and reconcile semantic conflicts among heterogenous sources. A context is a collection of statements that defines the properties and types of objects of interest. The architecture of the system consists of five main components: *a domain model*, *a context mediator*, *context axioms*, *elevation axioms wrappers*, *an optimizer*, and *an executioner*. These components are organized as follows. Each data source has a wrapper which allows to retrieve data from the source like previous systems. Furthermore, each source has a set of evaluation axioms, and a set of context axioms, and both converge to a common data model. The role of these components can be compared to that of the ontologies. The domain model contains a collection of definitions for source's primitive types (integer, strings, etc.) and the types of information units (called *semantic types*) which provide

	Type of Architecture	Data Sources	Integration Strategy	Ontology Language	Ontology Approach	Semantic Conflicts
SIMS	mediator/ wrapper-based	databases	LAV	LOOM	single-based	schema-level
TAMBIS	mediator/ wrapper-based	databases + Web-sources	GAV	GRAIL	single-based	schema-level
Carnot	mediator/ wrapper-based	databases	GAV	CycL	single-based	schema-level
InfoSleuth	agent-based	databases + Web-sources	LAV	OKBC	multiple-based	schema-level
OBSERVER	agent-based	databases + Web-sources	LAV	CLASSIC	multiple-based	schema-level
COIN	agent-based	databases + Web-sources	LAV	CLASSIC	hybrid-based	data-level

Table 4.2: Features of Data Integration Systems

a common vocabulary of the application domain. Concepts such as *semantic objects*, *modifiers* and *conversion functions* provide semantics of data inside and across information sources. A semantic object is an instance of semantic types. It has a set of properties (modifiers) that serve as annotations to make semantics of different contexts explicit. The modifiers might have different values in different contexts. Examples of such modifiers are quality, precision, currency, etc. In the relational model, these modifiers can be modelled as meta-attributes of a given attribute. The evaluation axioms define the mappings between attributes in a source and semantic types in the domain model. The context axioms define alternative interpretations of semantic objects in different contexts. Each source has one context but several sources may share the same context. There are two types of context axioms: (1) axioms that define semantics of data in a given source in terms of the values assigned to modifiers, and (2) axioms that define how modifiers's values of a given semantic object are converted between different source's contexts (conversion functions) [GBMS99].

As described above, the COIN's architecture suits the hybrid ontology-based approach with regard to the classification in Section 4.2. The COIN framework is constructed on a deductive and object-oriented language (called COINL) of the family of F(rame) logic [KLW95], which combines both features of object-oriented and deductive data models. Users queries submitted to the COIN system are intercepted by the context mediator. The context mediator reformulates a user query into a mediated query. This mediation process is done by identifying and resolving potential semantic conflicts involved in the query using declarative knowledge of the application domain represented by the domain model, the elevation axioms, and the context axioms. The COIN optimizer transforms this mediated query into an optimized query plan. This plan will be executed by the executioner which dispatches sub-queries to individual systems, collates the results, undertakes conversions which may be necessary when data are exchanged between two sources, and returns the answers to the receiver.

Comparison with our approach. Table 4.2 summarizes the features of the data integration systems described in this section. Although our ontology based approach is conceived for database systems to deal with a single database, it can also be extended for data integration systems to deal with multiple (heterogeneous) databases. Nevertheless, there are some similarities with the approaches described previously. Like OBSERVER, we use lexical relationships of types synonym, hypernym, and hyponym because these relationships allow to identify great similarities between concepts. However, OBSERVER use these relationships for describing the inter-connections between ontologies, but we use them for describing the ontology.

In most of the systems described above (e.g, SIMS, OBSERVER) a user should browse the ontology to find out the appropriate terms for formulating his query. Although this process assists users to formulate meaningful queries, it could be a tedious task, in particular, when the ontology is very large (i.e., contains a large amount of terms) or when the user is not familiar with the content of the ontology. In our approach, the user does not have to deal with ontologies directly, he can formulate his queries over the database as usual. It is the responsibility of the query processor to reformulate the query using mapping information about the database and its associated ontology. Therefore, our approach do not imply any change to the existing applications that access databases. This idea is similar to that used in Carnot where the CYC ontology is not used directly but it complements the semantics of the global schemas. However, in contrast to Carnot, in our approach the ontology is used at run time of query processing. That is, terms will be derived from the ontology for each query reformulation.

Another distinguishing feature of our approach is that mapping an ontology to its associated database takes into account both intensional (schema elements) and extensional information (data values) of the database content. The systems above (except COIN) either focus on ontology-schema matching or ontology-ontology matching. In COIN, mappings between data items is defined by converting the values from one source to others. In such systems, the ontology serves as a supporting role in the matching, but is not directly involved in the process. In our approach, linguistic information such as labels of concepts are used to specify ontology-data mappings.

4.6 Summary

In this chapter we presented the techniques used for query processing based on ontologies. We first explained the key role of ontologies to overcome problems encountered in query processing when dealing with different data semantics. Later, we presented the issues related to the mapping information utilized to link an ontology and a database. Indeed, we details the syntax and semantics of the different kinds of mappings that can be established between an ontology and a relational database. These mappings are necessary for processing queries by using semantic knowledge as we will see in the next chapters. In addition, we proposed algorithms that allow to find such mappings. Finally, we review the most relevant works related to our approach. We describe their features and their differences with respect to our proposal.

Chapter 5

Query Transformation Rules

This chapter presents a new approach to semantic query processing within single (object) relational database systems. The goal of the approach is to improve the quality of query answers using ontologies. For this purpose, we propose a set of semantic rules for transforming a user query into another more meaningful one, such that its results better meet the user's intention. In Sections 5.3 and 5.4 we classify the rules according to the result of the query transformation. We illustrate their usefulness by means of practical examples. In addition, in Section 5.5 we study basic properties for applying the rules based on a well-developed theory.

5.1 Approach

The objective of our approach to query processing is to reformulate an input query into another "meaningful" query, which might not necessary be equivalent to the original one. Our approach can be used in the context of existing DBMSs without substantial changes to their main components. The central idea is to transform a query in a preprocessing step before any query optimization. Figure 5.1 gives an overview of the proposed system's architecture.

The system mainly consists of three components. The first component is the transformation engine which constitutes the core of the system. It performs two kinds of query transformations: a *syntactic transformation* to rewrite an input query into a canonical form, followed by a *semantic transformation* to produce another query that better meet user's intention using semantic rules. These rules use semantic information extracted from an ontology. Basically, they

- Expand a user query by possibly changing its selection condition. This is done by extending query terms with other terms that have the same or a more specific meaning.
- Substitute the query condition with other conditions that are semantically equivalent.
- Reduce the scope of queries by restricting its context.

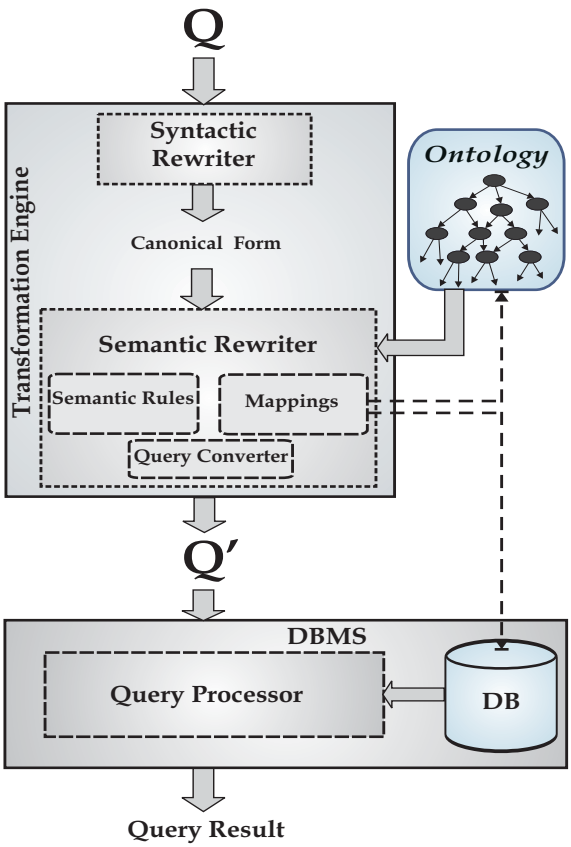


Figure 5.1: System Architecture

During query transformation, semantic rules are applied iteratively, in any order until no more transformations are possible. We define a transformation to be correct if the concepts and relationships corresponding to terms in Q and Q' are correctly represented in the ontology. Designing such rules is not an easy task. In this thesis, we develop a set of rules and illustrate their usefulness through practical examples. We believe that these rules can produce results closer to the user's expectations. The transformation uses mapping information between an ontology and a database. Under the assumption that the mappings are correct, as described in Chapter 4, any transformation carried out by a given rule is correct, i.e., implies a meaningful answer.

Moreover, we associate an ontology (second component) with the underlying database. The ontology could be either a general or a domain-oriented ontology depending on the nature of the database. Here, the role of the ontology is to provide additional semantics about the database content. We should note that the ontology should not be developed from scratch, but existing one could be adapted following our assumptions and requirements. That is, the ontology should completely cover the entities represented in the database and the mappings are correctly represented.

The third component is the DBMS which processes the output query and returns the answer to the user. The answer might contain more or less tuples than that answer produced by the original query. According to the size of the final result we classify the

transformation rules into two categories: *extension rules* and *reduction rules*. In the next sections we describe the rules in more detail and illustrate their usefulness using some practical examples.

We develop the rules for queries which have simple forms like queries that correspond to SPJ-queries [1] of the SQL-language.

Definition 5.1 (Query characteristics) *A query can be characterized by:*

- *A set of selection attributes A_Q.*
- *A set of relation names R_Q.*
- *A set of conditions (constraints) I_Q.*

We call all these characteristics the *query aspect* of Q. A_Q contains the output attributes corresponding to each value in a result's tuple. A_Q matches with the select list of an SQL-query. R_Q contains the basic relations used in the query. I_Q is conjunction of atomic predicates. An atomic predicate has the form: $u\theta v$, where u and v are constants or variables and θ is an arithmetic comparison operator ($=$, $<$ and so on). If u and v are both variables we refer to the corresponding condition as *join-predicate*. If u is a variable and v is a constant we refer to the corresponding condition as *selection-predicate*. In this thesis, we only deal with conditions whose selection predicates are equal predicates (i.e. θ is " = ") and that do not have negation.

5.2 Term Rewriting Systems

The theory of *term rewriting systems* is widely used as a computational model for rewriting queries [Fre85, DS89, Bry89, KM90]. In this section, we first introduce some basic notions of this theory as presented in [Hue80], then describe how it is applied to our transformation process. We use term rewriting systems to provide a sound basis for our query transformation and determine the fundamental properties for applying the rules.

5.2.1 Terms

Terms are built up from a finite set of function symbols $\mathcal{F}$, called *a signature*, and a set of variables $\mathcal{V}$ such that $\mathcal{F} \cap \mathcal{V} = \varnothing$. Each $f \in \mathcal{F}$ has a fixed arity $ar(f) \in \mathbb{N}$. We note that $c \in \mathcal{F}$ is called a *constant symbol* (or a constant) if $ar(c) = 0$. The set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ (or simply $\mathcal{T}$) of all terms over $\mathcal{V}$ is inductively defined as follows: $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for all $t_1, \ldots, t_n \in T(\mathcal{F}, \mathcal{V})$ and $f \in \mathcal{F}$ with $ar(f) = n$. A term having no variable is called *a ground term* and we denote the set of all ground terms by $T(\mathcal{F})$. Let t be a term of $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Let $\Omega(t) = \{t\} \bigcup_{i=1}^{n} \Omega(t_i)$ denotes the set of subterms in $t = f(t_1, \ldots, t_n)$ and let $V(t)$ denotes the set of variables occurring in

[1] SPJ: Select-Project-Join

t. Furthermore, we refer by $t[s_1/s_2]$ the term that is obtained from t by replacing a subterm s_1 in t by a term s_2.

A mapping σ from variables in $\mathcal{V}$ to terms in $\mathcal{T}$ is called a *substitution* if $\sigma(x) \neq x$ for only finitely many variables xs. Given a term t and a substitution σ, the application $\sigma(t)$ replaces all occurrences of variables in t by their respective σ-images. A term t is called *an instance* of a term s if there exists a substitution σ such that $\sigma(s) = t$. We may denote σ as $\sigma = \{x_1 \longmapsto \sigma(x_1), x_2 \longmapsto \sigma(x_2), \ldots\}$, where $x_i \in V(s)$. We refer to the set of all $\mathcal{T}$-substitutions as $\mathcal{S}(\mathcal{T})$. Any $\sigma \in \mathcal{S}$ can be extended to a matching substitution $\hat{\sigma}$ from $\mathcal{T}$ to $\mathcal{T}$ as follows: for $x \in \mathcal{V}$, $\hat{\sigma}(x) = \sigma(x)$ and for $s = f(x_1, \ldots, x_n) \in \mathcal{T}$, $\hat{\sigma}(s) = f(\hat{\sigma}(x_1), \ldots, \hat{\sigma}(x_n))$ [Hue80].

5.2.2 Principles of Term Rewriting Systems

A term rewriting system (TRS) is a pair $(\mathcal{T}, \mathcal{R})$ where $\mathcal{T}$ is a set of terms and $\mathcal{R}$ is a set of rewriting rules. A rewriting rule is a pair of terms (a, b), has the form $r : a \longrightarrow b$, such that $a \notin \mathcal{V}$ and $V(a) \supseteq V(b)$. We call a the *left-hand side* (*lhs*) and b the *right-hand side* (*rhs*) side of the rule. The application of the rule, written $\longrightarrow_r$ (or $\longrightarrow$ for many rules), transforms a term s into another term t by replacing an instance of the lhs in s with the corresponding instance of the rhs under a substitution σ. Formally

$$s \longrightarrow_r t \overset{def}{\iff} \exists\, \sigma \in \mathcal{S},\ s_0 \in \Omega(s) :\ \ s_0 = \sigma(a)\ \ and\ \ t = s[s_0/\sigma(b)]$$

We say that a term s *derives* another term t, denoted by $s \xrightarrow{*} t$, if t can be obtained by applying a finite sequence of rewrites using $\mathcal{R}$: $s \longrightarrow \ldots \longrightarrow t$. A term in the derivation sequence is called a *successor* of s. If this sequence constitutes of n steps t we denote it by $s \xrightarrow{n} t$. In this case we also say that the term s is *reducible*. If there is no term that could be derived from s then s is *irreducible*. If $s \xrightarrow{*} t$ and t is irreducible then we say that t is a *normal form* of s [BN98].

Definition 5.2 (Commuting Rules) *A rule r_1 commutes with a rule r_2, if for all terms x y_1, y_2 such that $y_1 \ {}_{r_1}\!\xleftarrow{*} x \xrightarrow{*}_{r_2} y_2$, there is term z such that $y_1 \xrightarrow{*}_{r_2} z$ and $y_2 \xrightarrow{*}_{r_1} z$.*

When dealing with term rewriting systems two important properties are frequently addressed, namely *Termination* and *Confluence*:

- Termination means that the rewriting process defined by the system stops during the application of rules. We say that the rewriting system is *notherian*. Formally, given a notherian TRS $(\mathcal{T}, \mathcal{R})$ and a term $t_i \in \mathcal{T}$, there is no infinite derivations $t_i \longrightarrow t_{i+1} \ldots$. That is, every term of $\mathcal{T}$ has at least one normal form.

- Confluence (also known as *Church-Rosser* property) $(\mathcal{T}, \mathcal{R})$ is *confluent* if the final result of a rewriting process does not depend on the application order of the rules. Formally, given a TRS $(\mathcal{T}, \mathcal{R})$, if there are different ways of applying $\mathcal{R}$ to a given term $t \in \mathcal{T}$ leading to different derived terms t_1 and t_2 of $\mathcal{T}$ then we can find a common term t_0 which can be derived both from t_1 and t_2 by applying the rules. Therefore, all pair of rules in $\mathcal{R}$ are commuting.

5.2.3 A Rewriting System for Queries

We use a TRS $(\mathcal{T}_e, \mathcal{R}_e)$ to transform formulas of the query expression into another one with respect to our approach. The language of terms $\mathcal{T}_e$ is the domain relational calculus language (DRC) [LP82]. The n-ary predicates expressed in queries are function symbols in $\mathcal{T}_e$. In addition, logical operators $\wedge$ and $\vee$ build binary function symbols which are written in infix form, i.e., we write $p_i \wedge p_j$ instead of $\wedge(p_i, p_j)$. $\mathcal{T}_e$ also includes (i) other logical connectors $\Rightarrow$ (implication), $\Leftrightarrow$ (equivalence), and (ii) logical quantifiers ($\exists$, $\forall$) or parentheses like '(' and '['. We define two types of variables for the TRS: (i) variables that range over domain variables of DRC formulas written in lower case and marked with macrons, e.g., $\bar{x}_i$, $\bar{y}_i$, and (ii) variables that range over DRC predicates written in upper case, e.g. $\bar{X}_i$, $\bar{Y}_i$. Constants are from the variable domains written in upper case, e.g., T_0, T_1. In addition, we define a substitution σ over terms of $\mathcal{T}_e$ as:

1. A mapping from variables $\bar{x}_i$ to DRC variables $\bar{x}_i$, e.g., $\{\bar{x}_1 \longmapsto x_1, \bar{x}_2 \longmapsto x_2, \ldots\}$.
2. A mapping from variables $\bar{X}_i$ to DRC predicates p_i, e.g., $\{\bar{X}_1 \longmapsto p_1, \bar{X}_2 \longmapsto p_2, \ldots\}$.

We extend the rule definition by introducing *conditional rules*. A conditional rule in $\mathcal{R}_e$ has the form $r : a \xrightarrow{\mathcal{C}} b$, which is applied whenever a condition $\mathcal{C}$ evaluates to *true*. The condition $\mathcal{C}$ is declarative and restricts the queried data. That is, it specifies properties of query objects (relations, attributes, values, etc.) that are referenced by terms in the expressions a and b. An example of such condition is asserting the existence of a mapping between a relation name and a concept in the ontology.

For our approach we define a query transformation using TRS as follows. A query Q can be transformed into query Q' by applying a given rule r if there exists a subterm in the formula of Q that matches the rule's lhs and satisfies the corresponding condition. The rewriting is iteratively done by substituting such subterm by the rule's rhs in each iteration. Q' is obtained at an iteration step n after which no further rewriting is needed, i.e., new queries will be equivalent. We denote this application by $Q \xrightarrow{*}_r Q'$. We should mention that during some iterations we may need additional rewriting for supporting the application of r (e.g., rearrange the order of predicates), as will be explained in Section 5.5. Formally, we define a rule application as follows.

Definition 5.3 (Rule Application) *Given a rule $r \in \mathcal{R}_e$ and query Q, such that $r : a \xrightarrow{\mathcal{C}} b$ and $Q = \{(x_1, \ldots, x_n) \mid F(x_1, \ldots, x_n)\}$ where F is a wff. The application of r to Q is defined as:*

$$Q \xrightarrow{*}_r Q' \overset{def}{\iff} \exists\, n \in \mathbb{N}\ F \xrightarrow{n}_r F_n \text{ such that } \forall\, i \leq n\ \mathcal{C} \equiv true \text{ and } F_i \longrightarrow_r F_{i+1}$$
$$\forall\, j \geq n\ F_j \equiv F_n$$
$$\text{and } Q' = \{(x_1, \ldots, x_n) \mid F_n(x_1, \ldots, x_n)\}.$$

In the next sections we describe the rules set $\mathcal{R}_e$ in more details and illustrate how they could be applied to transform queries.

5.3 The Set of Extension Rules

Extension rules aim at extending the query answer with results that meet user's expectations. To this end, we have developed fourteen rules grouped into six classes: SYNONYMY rule, COLLECTION rule, PART-WHOLE rules, SUPPORT rules, FEATURE rules, and CONSISTENCY rules. Briefly, SYNONYMY and COLLECTION rules are based on interpreting lexical terms that are used in user queries. PART-WHOLE rules are based on the Part-whole meaning of objects to identify their instances in the database. SUPPORT rules complement the PART-WHOLE rules and infers more knowledge about the objects. FEATURE rules use domain-specific relationships from an ontology for specifying queries. Finally, CONSISTENCY rules derive new constraints from the ontology that are semantically equivalent to those in the queries and use them in user queries. In the following, we describe these rules in more detail.

5.3.1 Synonymy Rule

The SYNONYMY rule addresses the problem of semantic ambiguities motivated in the problem definition. Basically, the rule bridges the semantic gap of the vocabularies used in the user's query and those stored in the database. Intuitively, the rule reformulates the query condition using terms derived from a given ontology which have the same meaning as query terms. This is done by following `SynOf`-relationships.

In the following, we first present the syntactical expression of the rule and then illustrate its application. We use predicates $\Psi^{A_i}_{\zeta\mathcal{D}}$, $\Psi^{*}_{\zeta\mathcal{U}}$ to respectively represent the mapping between concepts and domain values of attribute A_i , and the mapping between concepts and attributes, as described in Chapter 4, respectively.

Definition 5.4 (Synonymous Values) *Let $c \in \zeta$. We define the values in a database domain D which correspond to synonyms of c, i.e., $SYNs(c)$ [2] by the set $\Delta^1_c = \{v \in D \mid \exists\, s \in \zeta :\ \Psi_{\zeta\mathcal{D}}(s, v)\ \wedge\ Synof(s, c)\}$.*

Rule Definition. Given a relations $R \in \Sigma$ such that $R(A_1, \ldots, A_n)$, a string $T_0 \in D$ and two concepts $C_0, C_A \in \zeta$. Formally, the SYNONYMY rule is defined as follows:

Synonymy Rule:

$$[\, R(\bar{x}_1, \ldots, \bar{x}_n)\ \wedge\ \bar{x}_i = T_0\,] \xrightarrow{\mathcal{C}_1} [\, R(\bar{x}_1, \ldots, \bar{x}_n)\ \wedge\ (\bar{x}_i = T_0\ \vee \bar{x}_i = T_k)\,]$$

with condition $\mathcal{C}_1 = [\Psi^{*}_{\zeta\mathcal{U}}(C_A, A_i) \wedge \Psi^{A_i}_{\zeta\mathcal{D}}(C_0, T_0)]$ *and* $T_k \in \Delta^1_{C_0}$. □

The SYNONYMY rule extends the query conditions of the form $x_i = T_0$ using synonyms of T_0. To apply this rule all values of A_i including T_0 must be mapped to concepts in the ontology. Therefore, the extension is based on the mapping $\Psi^{A_i}_{\zeta\mathcal{D}}$ that links concepts to values of attribute A_i. These terms are given by the set $\Delta^1_{C_0}$ and are used for constructing additional disjunctions.

[2] See definition in Section 3.6

Example 5.1 Going back to example 3.1 of the problem definition, we assume that the product ontology PO (see Figure 3.7) is associated with the electronics database PDB. We define then a mapping between concept PRODUCT and relation Item1, a mapping between concept ELECTRONIC and attribute Name, and a mapping between sub-concepts of ELECTRONIC and domain values of Name. The value "computer" is mapped into concept COMPUTER. We suppose that a user wants to retrieve information about computers. He might submit a query to PDB as follows:

$$Q_0 = \{(x_1, x_2, x_3, x_4) \mid Item1(x_1, x_2, x_3, x_4) \wedge x_2 = \text{"}computer\text{"}\}.$$

Obviously, the answer to Q_0 contains the tuple identified by the id 123. However, according to the ontology PO the concept COMPUTER is synonymous with the concepts DATA PROCESSOR and CALCULATOR. Therefore, based on the mapping definitions the application of SYNONYMY rule to Q_0 will extend the query condition with disjunctions using all values T_k that have the same meaning as the value T_0= "computer" . That is, $T_k \in \Delta^1_{\texttt{COMPUTER}} = \{\text{"}calculator\text{"}, \text{"}data\ processor\text{"}\}$. The resulting query is expressed as follows:

$$Q_0' = \{(x_1, x_2, x_3, x_4) \mid Item1(x_1, x_2, x_3, x_4) \wedge (x_2 = \text{"}computer\text{"} \vee x_2 = \text{"}calculator\text{"} \vee x_2 = \text{"}data\ processor\text{"})\}.$$

We emphasize that the answer to the transformed query □

5.3.2 Collection Rule

The COLLECTION rule complements the SYNONYMY rule. It also addresses the problem of semantic ambiguities motivated in the problem definition. Intuitively, the rule reformulates the query condition using terms derived from a given ontology which have very similar meaning as query terms. This is done by substituting query terms with other terms which are more specific. The rule relies on the use of ISA-relationships from the ontology.

In the following, we first present the syntactical expression of the rule and then illustrate its application. We use predicates $\Psi^{A_i}_{\zeta\mathcal{D}}$, $\Psi^{*}_{\zeta\mathcal{U}}$ to respectively represent the mapping between concepts and domain values of attribute A_i , and the mapping between concepts and attributes, as described in Chapter 4.

Definition 5.5 (Specific Values) *Let $c \in \zeta$. We define the values of a database domain D which correspond to sub-concepts of c, i.e., $DESCs(c)$ [3] by the set $\Delta^2_c = \{v \in D \mid \exists\, s \in \zeta :\ \Psi_{\zeta\mathcal{D}}(s, v) \wedge Isa(s, c)\}$.*

Rule Definition. Given a relations $R \in \Sigma$ such that $R(A_1, \ldots, A_n)$, a string $T_0 \in D$ and two concepts $C_0, C_A \in \zeta$. Formally, the COLLECTION RULE is formulated as follows:

[3] See definition in Section 3.6

Collection Rule:

$$[\, R(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0 \,] \xrightarrow{\mathcal{C}_2} [\, R(\bar{x}_1, \ldots, \bar{x}_n) \wedge (\bar{x}_i = T_0 \vee \bar{x}_i = T_k) \,]$$

with condition $\mathcal{C}_2 = [\Psi^*_{\zeta\mathcal{U}}(C_A, A_i) \wedge \Psi^{A_i}_{\zeta\mathcal{D}}(C_o, T_o)]$ *and* $T_k \in \Delta^2_{C_o}$. □

The COLLECTION rule becomes applicable under the same condition as the SYNONYMY rule. The query extension is based on sub-concepts of C_0 defined by $\Delta^2_{C_0}$ for introducing new disjunction predicates.

Example 5.2 We assume again that the product ontology `PO` is associated with the electronics database `PDB` and use the mappings between them as defined in the previous example. We also suppose that a user submits the query Q_0 to `PDB`. Obviously, the answer to Q_0 contains the tuple identified by the id 123. However, according to the ontology `PO` the concept `COMPUTER` has a broader meaning than the specialized concepts `NOTEBOOK` and `PALMTOP`. Intuitively, the `ISA`-relationship implies a strong similarity between a concept and its sub-concepts. Since the `ISA`-relationship is transitive, the same argument can be applied to further specializations, i.e., `MACPC` and `INTELPC`. Therefore, based on the mapping definitions the application of COLLECTION rule to Q_0 will extend the query condition with disjunctions using all values that have more specific meaning than value T_0= "computer". That is, $T_k \in \Delta^2_{\texttt{COMPUTER}} = \{$"*pc*", "*intelpc*", "*macpc*", "*notebook*", "*palmtop*"$\}$. The resulting query is expressed as follows:

$$Q_1' = \{(x_1, x_2, x_3, x_4) \mid Item1(x_1, x_2, x_3, x_4) \wedge (x_2 = \text{"}computer\text{"} \vee x_2 = \text{"}pc\text{"} \vee x_2 = \text{"}macpc\text{"} \vee x_2 = \text{"}intelpc\text{"} \vee x_2 = \text{"}notebook\text{"} \vee x_2 = \text{"}palmtop\text{"})\}.$$

We emphasize that the answer to the reformulated query Q_1' will include new tuples identified by ids 124, 127, and 140, which meet the user's intention. □

5.3.3 Part-Whole Rules

The basic idea of the PART-WHOLE rules is the use of the "part-whole" properties to discover new database objects which are closely related to those the given query returns. Based on the semantic relationship `"PartOf"` the rules rewrite a user query by substituting the query terms by other semantically equivalent ones. That is, the concepts corresponding to the substituted terms together with the `"PartOf"`-relationships specify the same entities corresponding to the original query terms. Thus, the same type of the objects specified in the query can be defined in another way by using an alternative set of terms [NF04]. In the following, we first present the syntactical expressions of the rules and then explain their applications.

Definition 5.6 (Parts Values) *Let $c \in \zeta$. We define the values in a database domain D which correspond to part concepts of c, i.e., $Rchild(PartOf, c)$* [4] *by the set Δ^3_c as follows: $\Delta^3_c = \{v \in D \mid \exists\, p \in \zeta :\ \Psi_{\zeta\mathcal{D}}(p, v) \wedge Partof(p, c)\}$.*

[4] See definition in Section 3.6

Rule Definition. Given three relations R_1, R_2, $R'_2 \in \Sigma$: $R_1(A_1, \ldots, A_n)$, $R_2(B_1, B_2, \ldots, B_s)$ and $R'_2(B'_1, B'_2)$ where A_1, B_1 and (B'_1, B'_2) are primary keys, and B_2, B'_1, B'_2 are foreign keys referring to R_1. Moreover, given a string $T_0 \in D$ and two concepts $C_0, C_A, C_B \in \zeta$. We distinguish three rules according to how the relationship `"PartOf"` is mapped to the underlying database:

a) **Base Part-Whole Rule** :

$$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0 \,] \xrightarrow{\mathcal{C}_3} [\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge (\bar{x}_i = T_0 \bigvee_{k=1}^{m} \bar{x}_j = T_k) \,]$$

with condition $\mathcal{C}_3 = M_1 \wedge I_o$ □

b) **<1-n>Part-Whole Rule** :

$$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0 \,] \xrightarrow{\mathcal{C}_4} [\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0 \vee \bigwedge_{k=1}^{m} \exists\, (\bar{y}_{1k}, ..., \bar{y}_{sk}) \mid R_2(\bar{y}_{1k}, ..., \bar{y}_{sk}) \wedge \bar{x}_1 = \bar{y}_{2k} \wedge \bar{y}_{jk} = T_k \,]$$

with condition $\mathcal{C}_4 = M_2 \wedge I_o$ □

c) **<n-m>Part-Whole Rule**:

$$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0 \,] \xrightarrow{\mathcal{C}_5} [\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0 \vee \bigwedge_{k=1}^{m} \exists\, (\bar{y}_{1k}, \bar{y}_{2k})\ (\bar{z}_{1k}, \ldots, \bar{z}_{nk}) \mid R'_2(\bar{y}_{1k}, \bar{y}_{2k}) \wedge R_1(\bar{z}_{1k}, \ldots, \bar{z}_{nm}) \wedge \bar{x}_1 = \bar{y}_{1k} \wedge \bar{z}_{1k} = \bar{y}_{2k} \wedge \bar{z}_{ik} = T_k \,]$$

with condition $\mathcal{C}_5 = M_3 \wedge I_o$ and $T_k \in \Delta^3_{C_o}$, $m = |\Delta^3_{C_o}|$, *and* $s > 2$. □

Furthermore, I_0 is a formula that declares a constraint on the part concepts of C_0. It consists of predicates representing the `ISA` and `PartOf` relationships.

$$I_o : \forall\, p\ c \in \zeta\ [\ Partof(p, C_o) \wedge Partof(p, c) \wedge \neg Isa(c, C_o)\ ] \implies \exists\, p'\ [\ (Partof(p', c) \wedge \neg Partof(p', C_0)) \vee (Partof(p', C_0) \wedge \neg Partof(p', c))\ ]$$

We define M_1, M_2, and M_3 as formulas that declare mapping constraints on relation R_1 and some of its attributes. Each formula consists of predicates representing the mapping between concepts and relations $\Psi^*_{\zeta\Sigma}$, concepts and attributes $\Psi^*_{\zeta\mathcal{U}}$, concepts and database values $\Psi_{\zeta\mathcal{D}}$, relationships and attribute pairs $\Psi_{\Re\mathcal{P}}$, relationships and key attributes $\Psi_{\Re\mathcal{F}}$, and relationships and relations $\Psi_{\Re\Sigma}$, as described in Chapter 4. They are defined as follows:

$$M_1 :\ M_0 \wedge \Psi^*_{\zeta\mathcal{U}}(C_A, A_j) \wedge \Psi_{\Re\mathcal{P}}(PartOf, [A_i, A_j])$$
$$M_2 :\ M_0 \wedge \Psi^*_{\zeta\mathcal{U}}(C_B, B_j) \wedge \Psi_{\Re\mathcal{F}}(PartOf, B_2)$$
$$M_3 :\ M_0 \wedge \Psi_{\Re\Sigma}(PartOf, R'_2)$$

where M_0 is a common formula of the form $[\Psi^*_{\zeta\Sigma}(C_R, R_1) \wedge \Psi^*_{\zeta\mathcal{U}}(C_A, A_i) \wedge \Psi^{A_i}_{\zeta\mathcal{D}}(C_0, T_0)]$.

The BASE PART-WHOLE rule concerns queries that involve a single relation. In this case in which the "PartOf"-relationship is mapped to a pair of attributes of a *strong* relation R_1. There, each object represented in R_1 is associated with one part. However, <1-N>PART-WHOLE rule deals with the case in which the "PartOf"-relationship is mapped to a foreign key B_2 of a relation R_2 referencing a *strong* relation R_1. In addition, there is an attribute B_j which is mapped to a concept C_B in the ontology. In this case, each object represented in R_2 might be part of at most one object of R_1 whereas each object represented in R_1 might contain many part-objects of R_1 (see Section 4.3). However, the <N-M>-PART-WHOLE rule deals with the case in which the "PartOf"-relationship is mapped to a *combined* relation R'_2. There, each object represented in R_1 might be part (or may contain) many other objects. Obviously, we can formulate another rule expression similar to the later expression if the parts and the wholes are represented in two different relations.
In both cases the rules require that (i) there exists a mapping $\Psi^*_{\zeta\mathcal{U}}$ between an attribute A_i in R_1 and a concept C_A in the ontology, (ii) the query expression includes an equal predicate on A_i and a value T_0, and (iii) the parts of C_0 are *exclusive*, i.e., there are no other concepts, excepts its sub-concepts, that have all parts of C_0. Thus, the query Q is extended to a new query Q' in such way that Q' could also select tuples of the entities that contain all related parts. The database values T_k corresponding to these parts are defined by $\Delta^3_{C_0}$.

Example 5.3 We again assume that the product ontology PO is associated with the electronics database PDB and use the mappings between them as defined in the previous examples. We also assume that the value "pc" is mapped to concept PC. Now, we assume that a user wants to retrieve information about the Items 'pc' from the database PDB. His submitted query may look like

$$Q_2 = \{(x_1, x_2, x_3, x_4) \mid Item1(x_1, x_2, x_3, x_4) \ \wedge \ x_2 = \text{"}pc\text{"}\}.$$

Obviously, the answer to Q_2 contains one tuple identified by the id 127. However, if we examine the ontology PO we deduce that a PC is composed out of three parts: a desktop, a monitor, and a keyboard. Assuming that all PC-objects in the database are composed exactly out of these parts, which do not participate in the composition of any other object, enables the identification of PCs by means of their components. Thus, the set of terms $\{\text{"}desktop\text{"}, \text{"}monitor\text{"}, \text{"}keyboard\text{"}\}$ and the term "pc" are semantically equivalent. By applying the <N-M>PART-WHOLE rule to the query Q_2 we obtain a transformed query Q'_2 that retrieve also objects whose parts are the components given in S.

Formally, <N-M>PART-WHOLE rule has first to verify the assumption above and the mapping constraints. Afterward, it will extend the condition of Q_2 by introducing conjunctive predicates built up in terms of T_k, as defined in the rule's rhs. In this case, the values $T_k \in \Delta^3_{PC} = \{\text{"}monitor\text{"}, \text{"}keyboard\text{"}, \text{"}desktop\text{"}\}$. The query Q'_2 is expressed as follows:

$$Q'_2 = \{(x_1, x_2, x_3, x_4) \mid Item1(x_1, x_2, x_3, x_4) \wedge x_2 = \text{"}pc\text{"}\} \cup$$
$$\{(x_1, x_2, x_3, x_4) \mid Item1(x_1, x_2, x_3, x_4) \bigwedge_{k=1}^{3} \exists\, y_{1k}\ y_{2k}\, [\, Component(y_{1k}, y_{2k}) \wedge$$

$$x_1 = y_{1k} \wedge \exists\, z_{1k}\ z_{2k}\ z_{3k}\ z_{4k}\ [\, Item1(z_{1k}, z_{2k}, z_{3k}, z_{4k}) \wedge z_{1k} = y_{2k} \wedge z_{2k} = T_k\,]]\}$$

As results, it is surprising that tuples 123 and 128 with attribute values "computer" and "product" fully meet the intention of the user. When a user poses the query Q_2 to the PDB database, these tuples will certainly be missed. □

5.3.4 Support Rules

The basic idea of the SUPPORT rules is the use of structural information about objects by means of exploiting one of the intrinsic properties of the PartOf-relationship, namely the transitivity. In fact, by using the transitivity additional information about the components of an object can be deduced. This information could be useful for inferring more knowledge about database entities, hence characterizing them more precisely. We will illustrate this intuition in the example below.

Definition 5.7 (Values of Inferred Parts) *Let $c \in \zeta$. We define the values in a database domain D which correspond to inferred parts of c, i.e., $ANCES(PartOf, c)$* [5] *by the set $\Delta_c^4 = \{v \in D \mid \exists\, h \in \zeta\ : \Psi_{\zeta\mathcal{D}}(h, v) \wedge Partof(c, h)\}$.*

Rule Definition. Given three relations R_1, R_2, $R_2' \in \Sigma$: $R_1(A_1, \ldots, A_n)$, $R_2(B_1, B_2, \ldots, B_s)$, and $R_2'(B_1', B_2')$ where A_1, B_1 and (B_1', B_2') are primary keys, and B_1', B_2, B_2' are foreign keys referring to R_1. Moreover, given a string $T_0 \in D$ and four concepts $C_0, C_A, C_B, C_R \in \zeta$. In the following we distinguish three rules according to how "PartOf"-relationship is mapped onto the underlying database.

a) **Base-Support Rule** :

$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_j = T_0\,] \xrightarrow{\mathcal{C}_6} [\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge (\bar{x}_j = T_0 \vee \bar{x}_j = T_k)\,]$

with condition $\mathcal{C}_6 = M_o \wedge M_1$ □

b) **<1-n>Support Rule** :

$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge R_2(\bar{y}_1, \ldots, \bar{y}_s) \wedge \bar{x}_1 = \bar{y}_2 \wedge \bar{y}_j = T_0\,] \xrightarrow{\mathcal{C}_7}$
$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge R_2(\bar{y}_1, \ldots, \bar{y}_s) \wedge \bar{x}_1 = \bar{y}_2 \wedge (\bar{y}_j = T_0 \vee \bar{y}_j = T_k)\,]$

with condition $\mathcal{C}_7 = M_o \wedge M_2$ □

c) **<n-m>Support Rule**:

$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge R_2(\bar{y}_1, \bar{y}_2) \wedge R_1(\bar{z}_1, \ldots, \bar{z}_n) \wedge \bar{x}_1 = \bar{y}_1 \wedge \bar{z}_1 = \bar{y}_2 \wedge \bar{z}_i = T_0\,] \xrightarrow{\mathcal{C}_8}$
$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge R_2(\bar{y}_1, \bar{y}_2) \wedge R_1(\bar{z}_1, \ldots, \bar{z}_n) \wedge \bar{x}_1 = \bar{y}_1 \wedge \bar{z}_1 = \bar{y}_2 \wedge \bar{z}_i = T_0 \vee \bar{z}_i = T_k)\,]$

with condition $\mathcal{C}_8 = M_o \wedge M_3$
and $T_k \in \Delta_{C_o}^4,\ s > 2$.

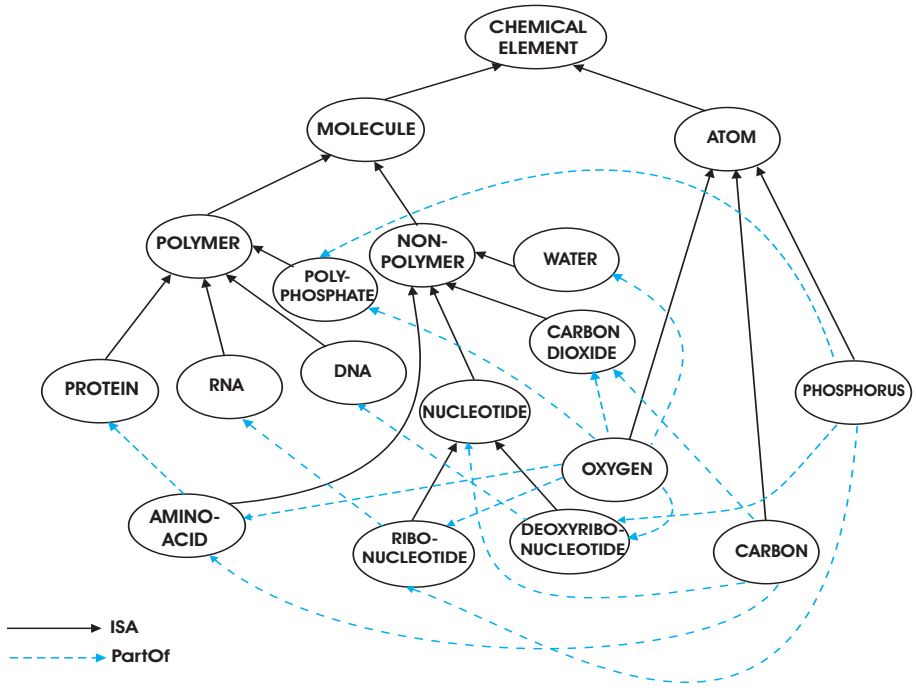


Figure 5.2: Biological Ontology (BO)

Furthermore, M_0, M_1, M_2, and M_3 are formulas that declare mapping constraints on relation R_1, R_2, and some of their attributes. Each formula consists of predicates representing the mapping between concepts and database values $\Psi_{\zeta\mathcal{D}}$, concepts and attributes $\Psi^*_{\zeta\mathcal{U}}$, relationships and attribute pairs $\Psi_{\Re\mathcal{P}}$, relationships and key attributes $\Psi_{\Re\mathcal{F}}$, relationships and relations $\Psi_{\Re\Sigma}$, and concepts and relations $\Psi^*_{\zeta\Sigma}$, as described in Chapter 4. They are defined as follows:

$$M_1 : \Psi^{A_j}_{\zeta\mathcal{D}}(C_0, T_0) \wedge \Psi^*_{\zeta\mathcal{U}}(C_A, A_i) \wedge \Psi_{\Re\mathcal{P}}(PartOf, [A_i, A_j])$$
$$M_2 : \Psi^{B_i}_{\zeta\mathcal{D}}(C_0, T_0) \wedge \Psi^*_{\zeta\mathcal{U}}(C_B, B_j) \wedge \Psi_{\Re\mathcal{F}}(PartOf, B_2)$$
$$M_3 : \Psi^{A_i}_{\zeta\mathcal{D}}(C_0, T_0) \wedge \Psi^*_{\zeta\mathcal{U}}(C_A, A_i) \wedge \Psi_{\Re\Sigma}(PartOf, R_2)$$
$$M_0 = \Psi^*_{\zeta\Sigma}(C_R, R_1)$$

On one hand, BASE-SUPPORT rule concerns queries that involve a single relation. The rule deals with the case in which the "PartOf"-relationship is mapped to a pair of attributes of a *strong* relation R_1. There, each object represented in R_1 is associated with one part. On the other hand, <1-N>SUPPORT and <N-M>SUPPORT rules concern queries that involve many relations. The former rule expression deals with the case in which the "PartOf"-relationship is mapped to a foreign key B_2 of a *combined* relation R_2 referring to R_1. There, each object represented in R_1 might contain many parts of R_2 whereas each object represented in R_2 might be part of at most one object of R_1 (see Section 4.3). However, the later expression deals with the case in which the "PartOf"-relationship is mapped to a *combined* relation R'_2. There, an object of R_1 might be part of many objects and vice versa. Obviously, we can derive from this

[5]See definition in Section 3.6

expression another rule expression if the parts and the wholes are represented in two different relations.

In all cases the rules require (i) a mapping of type $\Psi^*_{\zeta\mathcal{U}}$ between attributes and concepts implying correspondences between database values and concept in the ontology, (ii) that Q includes an equal predicate on one of such attributes and a value T_0, and (iii) T_k values from domain of that attribute for extending Q with disjunctions. These values are defined by $\Delta^4_{C_0}$.

Example 5.4 Suppose we have a database `BDB` containing entries about molecules. `BDB` has a relation, called `Compound`, which describes molecule compounds as shown in Figure 5.3. In addition, we suppose that we have a portion of a biological ontology, denoted by `BO` (see Figure 5.2). This ontology describes molecules and their components.

Compound(CID, Name, Component)
CID: Compound identifier
Name: Molecule name
Component: Compound component
PKey(CID)

CID	Name	Component
10	carbon dioxide	carbon
20	protein	amino-acid
30	enzyme	amino-acid
40	dna	nucleotide
50	water	hydrogen
60	rna	nucleotide
70	methane	carbon
80	ethane	carbon

Figure 5.3: Compound relation

We define a mapping between concept `CHEMICAL ELEMENT` and relation `Compound`, a mapping between concept `MOLECULE` and attribute `Name`, and a mapping between sub-concepts of `MOLECULE` and domain values of `Name`. The value "carbon" is then mapped into concept `CARBON`.

We suppose now that a user asks for information about all molecular compounds that contain carbons using `BDB`. His query can be represented as follows:

$$Q_3 = \{x_2 \mid \exists\, x_1\, x_3\; Compound(x_1, x_2, x_3) \wedge x_3 = \text{"}carbon\text{"}\}.$$

By submitting the query Q_3 to `BDB`, the DBMS returns three tuples including the carbon dioxide, the methane and the ethane compound names. However, if we investigate the ontology `BO`, we can deduce that carbons are also contained in other molecules among which are: amino acid, carbon dioxide, ribonucleotide, and deoxyribonucleotide molecules.

Formally, the mapping specifications satisfy the condition of the Base-Support rule. Thus, this rule can be applied to query Q_3 and hence the query condition will be extended using equal predicates with disjunctions. These predicates are built over the values $T_k \in \Delta^4_{CARBON} = \{$"*protein*", "*carbon dioxide*", "*nucleotide*", "*amino acid*", "*dna*", "*rna*"$\}$. The concepts related to these values are inferred from the concept `CARBON` based on the transitivity of the relationship `PartOf`. A user might not be aware of such information. The transformed query is expressed as follows:

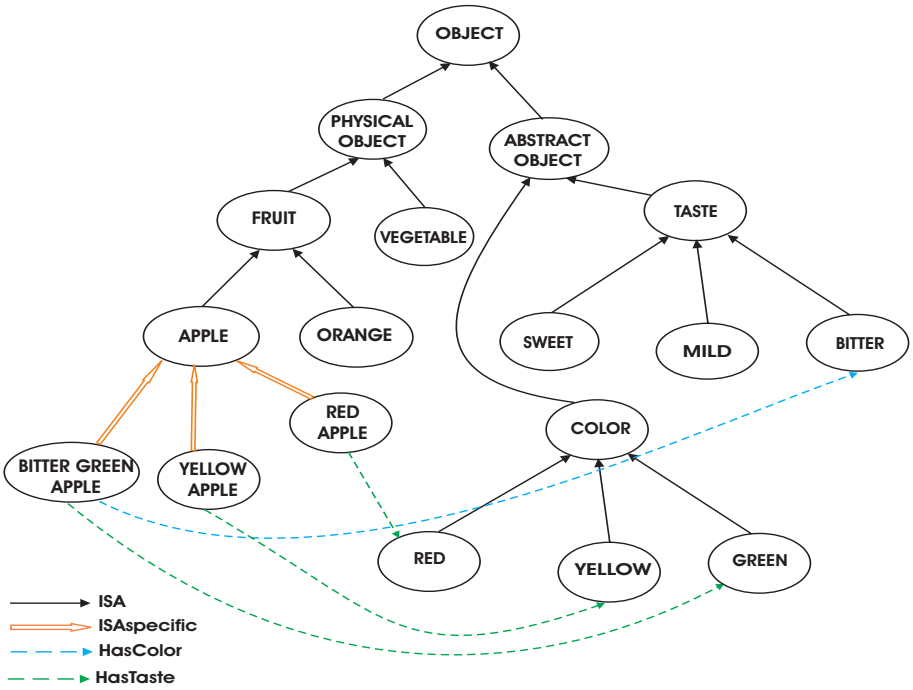


Figure 5.4: Fruit Ontology (FO)

$Q_3' = \{x_2 \mid \exists\ x_1\, x_3\ \mathit{Compound}(x_1, x_2, x_3) \wedge (x_3 = \text{"}\mathit{carbon}\text{"} \vee x_3 = \text{"}\mathit{protein}\text{"} \vee x_3 = \text{"}\mathit{carbon\ dioxide}\text{"} \vee x_3 = \text{"}\mathit{nucleotide}\text{"} \vee x_3 = \text{"}\mathit{amino\ acid}\text{"} \vee x_3 = \text{"}\mathit{dna}\text{"} \vee x_3 = \text{"}\mathit{rna}\text{"})\}$.

Surprisingly, the answer to Q_3' includes four additional tuples related to RNA, DNA, enzyme, and protein compound names, i.e., 20, 30, 40, and 60. These tuples are semantically correct and can meet the intention of the user. □

5.3.5 Feature Rules

Ontologies related to specific domains like medicine, geography use relationships specific to those domains. For instance, an ontology for describing fruits may contain domain-specific relationships like "HasColor" and "HasTaste". Figure 5.4 shows a portion of such ontology called FO. The "HasColor"-relationship means that if two concepts, say A and B, have a relationship of type "HasColor", then the objects referred by A have the color referred by B. Furthermore, another type of relationship "ISAspecific" is introduced. This type is similar to "ISA" but with additional restrictions for the related concept's objects. In fact, "ISAspecific" [6] is always combined with one or more domain-specific relationships when it is used to describe concepts. If a concept A subsumes a concept B through "ISAspecific" then the objects represented by A and B are distinguished only by a set of features specified by means of the domain-specific relationships associated with B. For example, the objects referred by the concept

[6]The corresponding predicate is denoted by $Isasp$

`RED APPLE` are more specified than those referred by `APPLE` since their color can be determined "red".

FEATURE rules exploit domain-specific relationships of the ontology to take advantages of additional features about entities represented by database instances. In fact, these features could be derived from the ontology and used to rewrite user queries. In the following we first present formal definitions of the rules and then give an illustrative example.

Rule Definition. Given a relation $R_1 \in \Sigma$ such that $R_1(A_1, \ldots, A_n)$ where A_1 is its primary key. Moreover, let $\Sigma_0 \subset \Sigma$ be a set of relations referred by R_1. That is, for each $R_h(B_{h1}, \ldots, B_{hs})$ ($R_h \in \Sigma_0$) there exists an attribute $A_l \in R_1$ which is its foreign key. We denote the set of these foreign keys Ω and the set of domain-specific relationships $\Re_0$ ($\Re_0 \subset \Re$). In addition, let two strings $T_0, T_q \in D$ and four concepts $C_0, C_A, C_q, C_R \in \zeta$.

For the rules definition we use predicates representing the mapping between concepts and relations $\Psi^*_{\zeta\Sigma}$, concepts and attributes $\Psi^*_{\zeta\mathcal{U}}$, concepts and database values $\Psi_{\zeta\mathcal{D}}$, relationships and attribute pairs $\Psi_{\Re\mathcal{P}}$, and relationships and key attributes $\Psi_{\Re\mathcal{F}}$, as described in Chapter 4. We distinguish two rule expressions according to how domain-specific relationships are mapped to the underlying database:

a) **Base-Feature Rule**:

$[R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0] \xrightarrow{\mathcal{C}_9} [R_1(\bar{x}_1, ..., \bar{x}_n) \wedge \bar{x}_i = T_0 \vee (\bar{x}_i = T_q \wedge \bar{x}_k = T_{0k})]$

with condition $\mathcal{C}_9 = M_0 \wedge Isasp(C_0, C_q) \wedge$
$\exists U \subset R_1 \forall A_k \in U \; \exists \beta_k \in \Re_0, \; c_{0k}\, c_k \in \zeta \; [M_1 \wedge Isa(c_{0k}, c_k)]$
and $T_{0k} \in \Delta^5_1 = \{v \in D \,|\, \beta_k(C_0, c_{0k}) \wedge \Psi^{A_k}_{\zeta\mathcal{D}}(c_{0k}, v)\}$ *and* $k \neq i$. □

b) **<1-n>Feature Rule** :

$[R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0] \xrightarrow{\mathcal{C}_{10}} [R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_i = T_0 \vee (\bar{x}_i = T_q \wedge \exists (\bar{y}_{h1}, ..., \bar{y}_{hs}) | \; R_h(\bar{y}_{h1}, ..., \bar{y}_{hs}) \wedge \bar{x}_1 = \bar{y}_{h1} \wedge \bar{y}_{h2} = T_h)]$

with condition $\mathcal{C}_{10} = M_0 \wedge Isasp(C_0, C_q) \wedge$
$\forall A_l \in \Omega \; \exists R_h \in \Sigma_0 \; \beta_h \in \Re_0, \; c_{0h}\, c_h\, c_{R_h} \in \zeta \; [M_2 \wedge Isa(c_{0h}, c_h)]$
and $T_h \in \Delta^5_2 = \{v \in D \,|\, \beta_h(C_0, c_{0h}) \wedge \Psi^{B_{h2}}_{\zeta\mathcal{D}}(c_{0h}, v)\}$. □

The values described in Δ^5_1 and Δ^5_2 correspond to concepts in the ontology which are related to C_0 through domain specific relationships. In addition, M_0, M_1, and M_2 are formulas that declare mapping constraints on relation R_1 and some of its attributes. They are defined as follows:

$M_0: \; \Psi^*_{\zeta\Sigma}(C_R, R_1) \wedge \Psi^*_{\zeta\mathcal{U}}(C_A, A_i) \wedge \Psi^{A_i}_{\zeta\mathcal{D}}(C_0, T_0) \wedge \Psi^{A_i}_{\zeta\mathcal{D}}(C_q, T_q)$
$M_1: \; \Psi^*_{\zeta\mathcal{U}}(c_k, A_k) \wedge \Psi_{\Re\mathcal{P}}(\beta_k, [A_i, A_k])$
$M_2: \; \Psi_{\Re\mathcal{F}}(\beta_h, A_l) \wedge \Psi^*_{\zeta\mathcal{U}}(c_h, B_{h2}) \wedge \Psi^*_{\zeta\Sigma}(c_{R_h}, R_h)$

Both rules state the following conditions on query terms for rewriting a query Q. They require that (i) Q contains an equal predicate on the attribute A_i and a value T_0, (ii) there are mappings between A_i and R_1 and concepts in ζ, and (ii) there are mappings between T_0 and concept C_0, and between T_q and concept C_q specified by C_0. However, the rules state different conditions on the mapping of relationships from $\Re_0$. On one hand, BASE-FEATURE rule deals with the case in which each relationship β_k in $\Re_0$ is mapped to a pair of attributes in R_1. In this case, there is a set of attributes U in R_1 which participate in the relationship mappings. Each attribute pair is composed from A_i and an attribute from U. All attributes A_k in U are mapped to concepts c_k in ζ. That is, each attribute pair correspond to a relationship β_k. In addition, the rule requires the existence of a concept c_{0k} which is a sub-concept of c_k and is related to C_q through β_k. Thus, database values T_q and T_{0k} which correspond to these concepts are considered for extending Q. The extension is done by substituting the original predicate with conjunctive predicates on attributes A_k, T_q, and T_{0k}. On the other hand, <1-N>FEATURE rule deals with the case in which there exists some relations R_h in Σ_0 that convey the semantic of some domain-specific relationships. That is, each relationship β_h in $\Re_0$ is mapped to a foreign key of relation R_h. Moreover, an attribute B_{h2} of R_h is mapped to concept c_h in ζ. That is, concepts representing values of A_i and B_{h2} are related through β_h. Thus, the values T_h, which correspond to concepts c_{0h} related to C_0, are used for extending Q with conjunctions on B_{h2}. Unlike SUPPORT rules FEATURE rules do not include a rule expression for the case in which there are many-to-many correspondences between objects represented in relations (rule of type <N-M>). This is due to the fact that if a given concept participates in certain domain-specific relationship then it is related to only one concept through that relationship (see Section 3.6).

Example 5.5 Assuming we have a database `FDB` of a store, which contains entries about some fruit goods. The `FDB`-schema has a relation, called `Goods`, which includes the name of each good, its color and its quantity (in kg). An instance of `FDB` and a description of the relation `Goods` are given in Figure 5.5. Moreover, we associate the ontology `FO` with the database `FDB` in order to describe its content. We define a mapping between concept `FRUIT` and relation `Goods`, a mapping between `FRUIT` and attribute `Name`, a mapping between sub-concepts of `FRUIT` and the domain values of `Name`, and a mapping between relationship `HasColor` and attribute pair (`Name`, `Color`). In addition, we assume that values "red-apple" and "apple" are mapped to concepts `RED-APPLE` and `APPLE`, respectively.

We suppose now that a user wants to retrieve all tuples from `FDB` concerning "red apple". His query can be represented as follows:

$$Q_4 = \{(x_1, x_2, x_3, x_4) \mid Goods(x_1, x_2, x_3, x_4) \wedge x_2 = \text{"}red\ apple\text{"}\}.$$

Obviously, the answer from the current `FDB` database to the Q_4-query contains only the tuple 16. However, according to the ontology `FO`, red apples are apples whose color is red.

Formally, having $T_0 =$ "*red apple*" and $T_q =$ "*apple*" the condition of BASE-FEATURE rule is satisfied. Thus, this rule can be applied to Q_4 and hence the query

Goods(<u>GID</u>, Name, Color, Quantity)
GID: Good identifier
Name: Good name
Color: Good color
Quantity: Quantity of good color
PKey(GID)

GID	Name	Color	Quantity
11	kiwi	green	30
12	banana	yellow	50
13	apple	red	70
14	cherries	red	20
15	peach	red	38
16	red-apple	red	52
17	plums	red	45

Figure 5.5: Goods relation

condition will be extended using two conjunctive predicates over values T_q and T_{01} ($T_{01} \in \Delta_1^5 = \{"red"\}$). The transformation of Q_4 will then result in the following query:

$$Q_4' = \{(x_1, x_2, x_3, x_4) \,|\, Goods(x_1, x_2, x_3, x_4) \wedge (x_2 = "red\ apple") \vee (x_2 = "apple" \wedge x_3 = "red")\}.\square$$

The query Q_4' provides more meaningful answer. In fact, it returns an additional tuple (tuple with id 13) which meets the user's intention.

5.3.6 Consistency Rules

The CONSISTENCY rules use the relationships between concepts related to a user query to derive other concepts that represent the same entities. The derived concepts will be used for transforming the query.

Before providing the syntax of the rules we first present some notions and terminologies which form the basis for the consistency rules. Furthermore, to illustrate the effectiveness of the rules we use the ontology `FmO`, called "Family Ontology", which describes concepts representing members of a family and the relationships among them. For clarity we provide a portion of this ontology as depicted in Figure 5.6. The complete description of `FmO` is given in Appendix A.1.1.

Atomic vs. derived Concepts/Relationships. Given an ontology, we distinguish between two kinds of concepts and relationships. A concept (or a relationship) is said to be *atomic* if it is not defined in terms of other concepts or relationships, otherwise, it is said to be *derived*. That is, if concepts and relationships are defined by axioms, for each derived concept (or relationship) say H, there is at least one axiom whose left-hand side is only H. Furthermore, atomic concepts (or relationships) do occur only on the right-hand side of axioms. We refer to concepts or relationships appearing in the right-hand side in an axiom of a derived concept or (a relationship) as being *constituents*. For instance, the concept `FATHER` in the family-ontology `FmO` is a defined concept since it is defined in terms of the concepts `PARENT` and `MAN` according to the axiom:

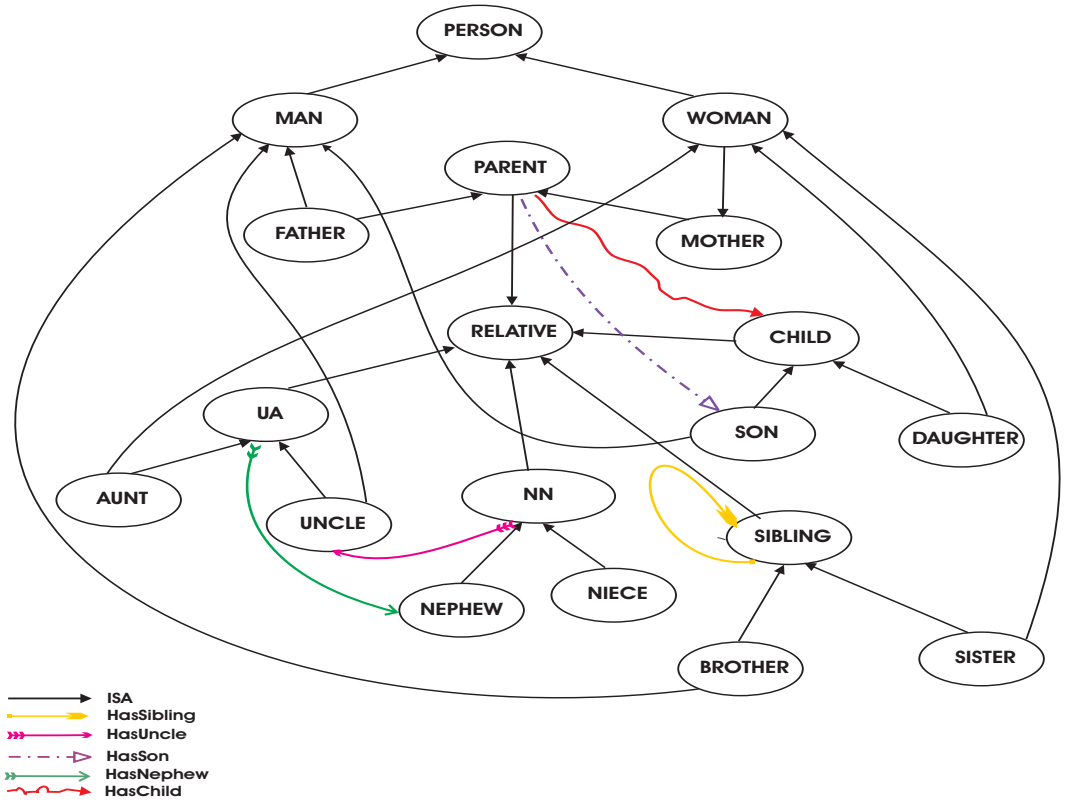


Figure 5.6: Family Ontology (FmO)

$$\forall x\ FATHER(x) \Longleftrightarrow PARENT(x) \wedge MAN(x) \qquad (1)$$

However, the concepts WOMAN and MAN are atomic (the only ones) since they do not appear in a left-hand side of any axiom (see Appendix A.1.1). On the other hand, according to the axiom

$$\forall\, x\, y\, \exists\, p\ HasUncle(x, y) \Longrightarrow HasParent(x, p) \wedge HasBrohter(p, y) \qquad (2)$$

the HasUncle-relationship is a derived relationship since it is expressed in terms of HasParent and HasBrother relationships. However, the only atomic relationship of FmO is the HasChild-relationship.

If a database represents instances of an atomic concept (or a relationship), this concept (or relationship) will be mapped to a base relation while if it represents instances of a derived concept (or relationship), this concept (or relationship) will be mapped to derived relations as seen in Section 4.3.1.

Definition 5.8 (Ontology-based Consistency) *A database is said to be* consistent *with respect to an ontology if the database instances representing ontology concepts satisfy all the axioms $\Im$ of the ontology.*

Moreover, if two axioms are equivalent then they must represent the same database instances. We define two axioms to be equivalent if they define the same derived concept or relationship (see in Example 5.7).

The execution of a query over a database extension returns a set of tuples. Usually, the tuples share common semantics by describing common features of the objects they represent. These features can be expressed by a set of conditions in the query. For instance, assume we have a database, denoted by FmDB, which contains information about individuals of a family. The schema of FmDB is described in Appendix A.1.2. Now suppose a user intends to request FmDB to get information about persons and their nephews. His query might look like

$$Q = \{(x_1, z_1) \mid Nephew(x_1) \wedge \exists\, y_1\, y_2\, [NephewOf(y_1, y_2) \wedge Uncle(z_1) \wedge x_1 = y_1 \wedge y_2 = z_1\,]\}$$

The evaluation of Q against FmDB-database returns instances that represent the *cousinship* (relationship of cousins) between uncles and nephews. In this context, semantics of a query could be specified using a set of conditions over a set of attributes of the relations used in that query. These semantics may be defined using an ontology associated with the database to which the query is posed. In fact, at the ontology level, query semantics might be captured by a fragment of the ontology, i.e., a set of particular concepts, the relationships that relate them, and their meanings (expressed by logical axioms). Thus, such a fragment represents also components of the query aspect in the ontology [NF05a].

Definition 5.9 (Semantic Vector) *Given a query Q, we define the* semantic vector *of Q, denoted ϖ_Q, as an ontology fragment described by the triple:*

$$\varpi_Q = \langle \zeta_Q, \Re_Q, \Im_Q \rangle$$

where ζ_Q is a subset of concepts from ζ, $\Re_Q$ is a subset of relationships from $\Re$ and $\Im_Q$ is a subset of axioms from $\Im$.

To determine ϖ_Q we need to map the components of the query aspect of Q to ontology elements. If the mapping is complete, i.e., all components are mapped, then we say that ϖ_Q completely covers the semantics of the query, otherwise, ϖ_Q partially covers its semantics. We often encounter the latter situation if I_Q contains selection-predicates. For instance, the semantic vector of Q can be defined as follows.

$$\varpi_Q = \langle \{UNCLE, NEPHEW\}, \{HasNephew\}, \{\forall\, x\, y \exists\, a\; HasNephew(x, y) \Longrightarrow HasSibling(x, a) \wedge HasSon(a, y)\}\rangle$$

We notice that ϖ_Q completely covers the semantics of Q. However, for example, if a user specifies nephews to whom the uncle should be related then ϖ_Q will partially cover the semantics of the query because there are no mappings of database values in FmO. For instance, such a query might be the following:

$$Q' = \{x_1 \mid Nephew(x_1) \wedge \exists\, y_1\, y_2\, [NephewOf(y_1, y_2) \wedge \exists\, z_1\, [Uncle(z_1) \wedge x_1 = y_1 \wedge y_2 = z_1 \wedge z_1 = 3]]\}$$

We also note that if the selection-condition changes, the semantic vector does not change since ontologies do not contain instances (w.r.t. our definition). Therefore,

the same fragment of O might cover multiple queries (i.e., mapped to multiple query aspects).

On the other hand, a given query might be covered by more than one fragment from the ontology representing equivalent semantics. That is, to a given query aspect (see Definition 5.1) we might assign more than one semantic vector. We define two semantic vectors to be equivalent if they represent the same concept entities. One of the major reasons for this multiple representations is that concept entities, to which tuple values are mapped, could belong to several concepts and might play several roles (by participating in different relationships). Thus, another equivalent semantics of a query could be derived from the ontology based on the initial semantics representation. For example, an individual, say "John", could be an individual of the concept `NEPHEW` and at the same time an individual of the concept `SON`. If so, then "John" has to participate in the relationships `"HasUncle"` or `"HasAunt"` and `"HasSon"`, i.e., he has an uncle, say "Smith", (or an aunt) and has a parent, say "Mark".

Each relationship invoked by the query must be investigated using information from the set $\Im$ of O in order to derive other equivalent semantic vectors. If a relationship is a derived relationship, it could be then interpreted in terms of its constituents. Constituent-relationships allow us to capture additional meanings about a derived relationship.

Example 5.6 The $\Im$ of the family-ontology `FmO` contains the following axiom:

$$\forall\, x\ y\ \exists\, a\ HasNephew(x, y) \Longrightarrow HasSibling(x, a) \wedge HasSon(a, y) \qquad (3)$$

This axiom indicates that if an individual x has a nephew y then it has a sibling a whose son is y. According to this axiom the relationship `"HasNephew"` is specified in terms of the `"HasSibling"`- and `"HasSon"` relationships. Therefore, "Smith", the uncle of "John", has a sibling who might be the parent "Mark" whose son is "John". Consequently, determining the nephews of an individual of `UNCLE` has the same meaning as determining the sons of his siblings. Therefore, all concepts and logical axioms related to constituent-relationships constitute the content of an additional semantic vector. For instance, another semantics of Q could be derived from `FmO` as defined by the following semantic vector:

$$\begin{aligned}\varpi_Q^{'} = \langle &\{SIBLING, PARENT, SON\},\ \{HasSibling, HasSon\},\\ &\{\forall x\ y\ HasSibling(x, y) \Longrightarrow HasSibling(y, x);\\ &\forall x\ y\ HasSon(x, y) \Longrightarrow HasChild(x, y) \wedge SON(y)\}\rangle\end{aligned}$$ □

In summary, mapping a given query (i.e., query aspect) to elements of the ontology (i.e., semantic vector) might allow us to derive another equivalent semantics, i.e., other ontology fragments that represent the same concept instances of its initial query semantics.

Use of Mappings. So far we have described how multiple semantics of a query can be extracted from an ontology. Now, we want to illustrate how to specify their corresponding instances in the database. For this purpose we use mapping information

Relationship	**Derived relation**
HasSibling	$\{(x,y) \mid \exists p\ Parenthood(p,x) \wedge Parenthood(p,y)\}$
HasNephew	$\{(x,y) \mid SiblingOf(x,y) \wedge \exists z,t\ [Parenthood(z,t) \wedge \exists u[Son(u) \wedge y=z \wedge t=u]]\}$
HasNiece	$\{(x,y) \mid SiblingOf(x,y) \wedge \exists z,t\ [Parenthood(z,t) \wedge \exists u[Daugther(u) \wedge y=z \wedge t=u]]\}$
HasUncle	$\{(x,y) \mid Parenthood(x,y) \wedge \exists z,t\ [SiblingOf(z,t) \wedge \exists u[Brother(u) \wedge y=z \wedge t=u]]\}$
HasAunt	$\{(x,y) \mid Parenthood(x,y) \wedge \exists z,t\ [SiblingOf(z,t) \wedge \exists u[Sister(u) \wedge y=z \wedge t=u]]\}$

Table 5.1: Mapping $\Psi'_{\Re\mathcal{E}}$ for `FmDB` and `FmO`

about derived relationships: We define for each relationship new mappings based on the mappings of its constituents (see Appendix A.1.3). This information can be provided by the axiom set $\Im$ of the ontology as illustrated by the following example:

Given the axiom (3), according to the mapping $\Psi_{\Re\mathcal{E}}$ between `FmO` and `FmDB` the relationships `HasSibling`- and `HasSon`-relationships are mapped to the relations expressed by $\{x \mid SiblingOf(x)\}$ and $\{(x,y) \mid Parenthood(x,y) \wedge \exists z\ [Son(z) \wedge y=z]\}$, respectively. The result of a join between these relations corresponds to the database instances to which the relationship `"HasNephew"` can be mapped. It can be expressed by $\{(x,y) \mid SiblingOf(x,y) \wedge \exists z\,t\ [Parenthood(z,t) \wedge \exists u\ [Son(u) \wedge y=z \wedge t=u]\}$. Consequently, database instances to which a derived relationship is mapped, could be determined by joining the relations corresponding to their constituents.
Let Δ_v be a set of DRC-expressions which define derived relations for each derived relationship of O. We define then a new mapping $\Psi'_{\Re\mathcal{E}}$ over the sets $\Re$ and Δ_v. For example, Table 5.1 depicts this mapping type concerning the database `FmDB` and its associated ontology `FmO`.

Rule Definition. Given a query Q, CONSISTENCY rules capture equivalent semantics of Q from the ontology, in order to reformulate Q into another query Q'. The query Q' could provide additional results that meet user's intention. Intuitively, the rules have been developed for those cases where there exist mappings between relationships of the ontology and relation names that appear in Q. More clearly, if Q invokes a relation (or a join between a set of relations) which expresses a semantic relationship between concepts representing these relations in the ontology, then Q can be transformed using a derived relation that corresponds to that relationship. The derived relation can be determined using additional mappings for that relationship. Consequently, after the transformation, if a database is not consistent with its associated ontology, the answer of Q' might contain more tuples than the answer of Q. The additional tuples would

interpret the semantics of the original query. In the following, we present two rules depending on the different types of the relationship mappings.

Given four relations R_1, R_2, R'_2, $R_3 \in \Sigma$ such that $R_1(A_1, \ldots, A_n)$, $R_2(B_1, \ldots, B_s)$, $R'_2(B'_1, B'_2)$, and $R_3(E_1, \ldots, E_m)$. The attributes A_1, B_1, E_1 and (B'_1, B'_2) are primary keys, and B_2, B'_2 are foreign keys referring to R_1. Let be also $T_0 \in D$, two concepts $C_1, C_2 \in \zeta$, and a relationship $\beta_0 \in \Re$.

<1-n>Consistency Rule:

$$[\, R_2(\bar{y}_1, \ldots, \bar{y}_n) \wedge R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge \bar{x}_1 = \bar{y}_2 \wedge \bar{x}_i = T_0 \,] \xrightarrow{\mathcal{C}_{11}} [\, R_2(\bar{y}_1, \ldots, \bar{y}_n) \wedge R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge (\exists\, \bar{w}_1\, \bar{w}_2 \mid V_0(\bar{w}_1, \bar{w}_2) \wedge \bar{y}_1 = \bar{w}_1 \wedge \bar{x}_1 = \bar{w}_2 \wedge \bar{x}_i = T_0) \,]$$

with condition $\mathcal{C}_{11} = M_0 \wedge M_1$. □

<n-m>Consistency Rule:

$$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge R'_2(\bar{y}_1, \bar{y}_2) \wedge R_3(\bar{z}_1, \ldots, \bar{z}_n) \wedge \bar{x}_1 = \bar{y}_1 \wedge \bar{y}_2 = \bar{z}_1 \wedge \bar{z}_i = T_0 \,] \xrightarrow{\mathcal{C}_{12}} [R_1(\bar{x}_1, \ldots, \bar{x}_n) \wedge R_3(\bar{z}_1, \ldots, \bar{z}_n) \wedge (\exists\, \bar{w}_1\, \bar{w}_2 \,|\, V_0(\bar{w}_1, \bar{w}_2) \wedge \bar{x}_1 = \bar{w}_2 \wedge \bar{z}_1 = \bar{w}_1 \wedge \bar{z}_i = T_0) \,]$$

with condition $\mathcal{C}_{12} = M_0 \wedge M_2$. □

The formulas M_0, M_1, and M_2 declare mapping constraints on relations R_1, R_2, and R_3 and some of their attributes. They include predicates representing the mapping between concepts and relations $\Psi_{\zeta\Sigma}$, and relationships and key attributes $\Psi_{\Re\mathcal{F}}$, as described in Chapter 4. In addition, the formula M_0 includes a predicate representing the mapping between relationships and derived relations $\Psi'_{\Re\mathcal{E}}$. These formula are defined as follows:

$$M_0 : \exists\, V_0 \in \Delta_v : \Psi'_{\Re\mathcal{E}}(\beta_0, V_0) \wedge \Psi_{\zeta\Sigma}(C_1, R_1)$$
$$M_1 : \Psi_{\Re\mathcal{F}}(\beta_0, B_2) \wedge \Psi_{\zeta\Sigma}(C_2, R_2)$$
$$M_2 : \Psi_{\Re\Sigma}(\beta_0, R'_2) \wedge \Psi_{\zeta\Sigma}(C_2, R_3)$$

<1-N>CONSISTENCY rule concerns the case in which user query Q invokes a mapping between a relationship in the ontology and an attribute in relation R_2. Indeed, there exists a mapping between a relationship β_0 and a foreign key B_2 referring to relation R_1. In addition, the concepts supporting β_0 are mapped to R_1 and R_2 (see Section 4.3). In this case, one object represented in R_1 might be related to many objects in R_2 but not vise versa. However, <N-M>CONSISTENCY rule concerns the case in which there exists a Relationship-Relation mapping $\Psi_{\Re\Sigma}$ between β_0 and a *combined* relation R'_2 invoked in Q. That is, β_0 specifies not only a possible correspondence between one object in R_1 and many others in R_3 but also a possible correspondence between one object in R_3 and many others in R_1 (see Section 4.3). Both rules require that the relationship β_0 should also be mapped to a derived relation V_0 as specified by $\Psi'_{\Re\mathcal{E}}$. The expression of V_0 is used for rewriting Q.

NEPid
1
2

(a) Nephew

NEPid	UAid
1	1
1	2
2	3

(b) NephewOf

Uid
1
2
3

(c) Uncle

Gid1	Gid2
1	4
1	5
2	4
3	6

(d) SiblingOf

Eid	Cid
2	1
4	3
4	1
5	4
6	2

(e) Parenthood

Kid
1
2
3
4
5

(f) Son

Figure 5.7: `FmDB` relations

Example 5.7 Assuming that database `FmDB` includes the instances described in Figure 5.7. We associate the database `FmDB` with the ontology `FmO`. We define mappings between the relationships and the database as described above. Moreover, we define one-to-one mappings between the concepts and the relation names such as mapping concept `UNCLE` to relation `Uncle`. Now, suppose that a user inquires `FmDB` for nephews whose uncle has id=1. The submitted query may look like:

$$Q_5 = \{x_1 \mid Nephew(x_1) \wedge \exists\, y_1\, y_2\, [NephewOf(y_1, y_2) \wedge \exists\, z_1\, [Uncle(z_1) \wedge x_1 = y_1 \wedge\ y_2 = z_1 \wedge\ z_1 = 1\ ]]\}$$

Obviously, the result of query Q_5 is the tuple 1. However, according to the mapping definitions (see Appendix A.1.3) the relation `NephewOf` is mapped into the relationship `HasNephew`. Thus, we deduce that that the query Q_5 invokes a relationship of type `HasNephew` between `Nephew` and `Uncle` entities. However, this relationship can also be mapped to a derived relation as specified by $\Psi'_{\Re\mathcal{E}}$ (see Table 5.1). Accordingly, the condition of <N-M>CONSISTENCY rule is satisfied and the rule can then be applied to Q_5. Hence, method<n-m>Consistency rule will transform Q_5 into the the following query:

$$Q'_5 = \{s \mid V_0(g, s)\ \wedge\ \exists\, z_1\, [Uncle(z_1) \wedge z_1 = g\ \wedge\ z_1 = 1]\}$$

where $V_0 = \{(g, s) \mid \exists\, g\, g'\, SiblingOf(g, g') \wedge \exists\, e\, c \quad [Parenthood(e, c) \wedge [Son(s) \wedge\ g' = e \wedge c = s\ ]]\}$.

The answer to query Q'_5 contains the tuples 1, 3, and 4. As a result, the user obtains additional tuples which meet his intention because the sons identified by 3 and 4 are nephews of the mentioned uncle according to the ontology semantics provided by `FmO`. Thus, in this state the `FmDB`-database is not consistent with respect to the `FmO`-ontology. □

5.4 The Set of Reduction Rules

Reduction rules aim at reducing the number of irrelevant tuples from the query answer. In the following, we describe one of such rules. We call this rule the SENSITIVITY RULE because its goal is to increase the *sensitivity* of a user query. A query is called *sensitive* if its answer contains as few as possible *false positives*. We define a tuple as *false positive* if it is semantically not correct w.r.t. the user's expectations.

5.4.1 Sensitivity Rules

A problem might occur when a database contains homonymous terms. If a user queries a database using terms in his query that are homonymous to some other terms in that database, the answer to his query might contain tuples that are irrelevant to him. For instance, the term "bank" has different meanings. It either means a container for keeping coins or a piece electronic device for saving data on or a container for saving money [Bee04]. Therefore, if a user queries a given database for information concerning an object "bank", the database might return tuples containing data about electronic device or containers or institutions of type bank. This might not meet the user's intention if the user expects data only on device.

To solve this problem, we propose transformation rules based on the use of an ontology associated with the given database. By applying this rule a context could be specified for a user query. That is, the context defined by the semantic description of the data, which uses vocabularies from the ontology to express the user's intention. The intuition is to specify user queries sufficiently to derive the relevant meaning based on the ontology concepts. Thus, in the example above, the user's intention to find information about "bank" as electronic device can be specified by domain specific ontologies which can describe different aspects of devices. That is, the context of user queries will be restricted to electronic devices [NF05b].

However, if the ontology is more general, i.e., specifies more than one context (e.g., `EnO`). In this case it would be difficult to determine the user's intention immediately. For example, the concept `BANK` might label two different nodes in two different subgraphs of the ontology. Each subgraph represents the related context of "bank". We suggest that the system asks the user to specify a unique "context". This could be done by providing him with the possibility to choose one of the ontology contexts in terms of the *immediate uncommon concepts* of the `BANK` concepts. The immediate uncommon concepts of two given concepts are defined in terms of the *least common concept* as follows.

Definition 5.10 (Least Common Concept) *Let a, b, l be concepts. l is a* least common concept *(lcc) of a and b iff*

- $a \in DESC(ISA, l)$ *and* $b \in DESC(ISA, l)$,
- $\forall\, k, k' \in \zeta$, *if* $a, b \in DESC(ISA, k) \cap DESC(ISA, k')$ *then* $k = k'$
- *if* $\exists\, c' \in \zeta \mid a \in DESC(ISA, c')$ *and* $b \in DESC(ISA, c')$ *then* $l \in DESC(ISA, c')$

Definition 5.11 (Immediate uncommon Concepts) *Let a, b, m, m', g, and g' be concepts. m and m' ($m \neq m'$) are* immediate uncommon concepts *(imuc) of a and b resp, iff*

- $\exists\, l \in \zeta \mid l = lcc(a, b)$ *AND*
- $m = RChild(ISA, l) \wedge m' = RChild(ISA, l)$

For example, the immediate uncommon concepts of the concepts `COMPUTER` and `MUSEUM` are the concepts `DEVICE` and `FACILITY`, respectively, since their least common concept is the concept `ARTIFACT`.

Rule Definition. Given a relation $R_1 \in \Sigma$ such that $R_1 = (A_1, \ldots, A_n)$ where A_1 is its primary key. Moreover, let $\Sigma_0 \subset \Sigma$ be a set of relations referred by R_1. That is, for each $R_h \in \Sigma_0$, $R_h = (B_{h1}, \ldots, B_{hs})$, there exists an attribute $A_l \in R_1$ which is its foreign key. We denote the set of domain-specific relationships $\Re_0$ ($\Re_0 \subset \Re$). In addition, let T_0 be a string in D and C_0, C_A, C_R be three concepts in ζ.

To define the rules we use predicates representing the mapping between concepts and database values $\Psi_{\zeta\mathcal{D}}$, concepts and attributes $\Psi^*_{\zeta\mathcal{U}}$, relationships and attribute pairs $\Psi_{\Re\mathcal{P}}$, relationships and key attributes $\Psi_{\Re\mathcal{F}}$, and concepts and relations $\Psi^*_{\zeta\Sigma}$, as described in Chapter 4. We distinguish two rules according to how domain-specific relationships are mapped to the underlying database:

a) **Base-Sensitivity Rule**:

$$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \;\wedge\; \bar{x}_i = T_0\,] \xrightarrow{\mathcal{C}_{13}} [\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \;\wedge\; \bar{x}_i = T_0 \;\wedge\; \bar{x}_k = T_{0k}\,]$$

with condition $\mathcal{C}_{13} = M_0 \;\wedge\; \exists\, c_k\, c_{0k} \in \zeta, \beta_k \in \Re_0\, [M_1 \;\wedge\; Isa(c_{0k}, c_k)]$
and with $T_{0k} \in \Delta^6_1 = \{v \in D \mid \exists\, c_{0k} \in \zeta\ [\beta_k(C_0, c_{0k}) \wedge \Psi_{\zeta\mathcal{D}}(c_{0k}, v)]\}$. □

b) **<1-n>Sensitivity Rule** :

$$[\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \;\wedge\; \bar{x}_i = T_0\,] \xrightarrow{\mathcal{C}_{14}} [\, R_1(\bar{x}_1, \ldots, \bar{x}_n) \;\wedge\; \bar{x}_i = T_0 \;\wedge\; \exists\, (\bar{y}_{h1}, \ldots, \bar{y}_{hs}) \mid R_h(\bar{y}_{h1}, \ldots, \bar{y}_{hs}) \wedge \bar{x}_1 = \bar{y}_{h1} \wedge \bar{y}_{h2} = T_{0h}\,]$$

with condition $\mathcal{C}_{13} = M_0 \wedge \exists\, \beta_h \in \Re_0, c_{0h}\, c_h\, c_{R_h} \in \zeta\, [M_2 \wedge Isa(c_{0h}, c_h)]$
and with $T_{0h} \in \Delta^6_2 = \{v \in D \,|\, \Psi^{B_{h2}}_{\zeta\mathcal{D}}(c_{0h}, v) \wedge\ \beta_h(C_0, c_{0h})\}$. □

The values in Δ^6_1 and Δ^6_2 correspond to concepts in the ontology that represent domain specific features. Furthermore, M_0, M_1, M_2, and M_3 are formulas that declare mapping constraints on relation R_1, R_h, and some of their attributes. They are defined as follows:

$$M_0:\ \Psi^*_{\zeta\Sigma}(C_R, R_1) \wedge \Psi^*_{\zeta\mathcal{U}}(C_A, A_i) \wedge \Psi^{A_i}_{\zeta\mathcal{D}}(C_0, T_0)$$
$$M_1:\ \Psi^*_{\zeta\mathcal{U}}(c_k, a_k) \;\wedge\; \Psi^*_{\Re\mathcal{P}}(\beta_k, [A_i, A_k])$$
$$M_2:\ \Psi_{\Re\mathcal{F}}(\beta_h, A_l) \wedge \Psi^*_{\zeta\mathcal{U}}(c_h, B_{h2}) \;\wedge\; \Psi^*_{\zeta\Sigma}(c_{R_h}, R_h)$$

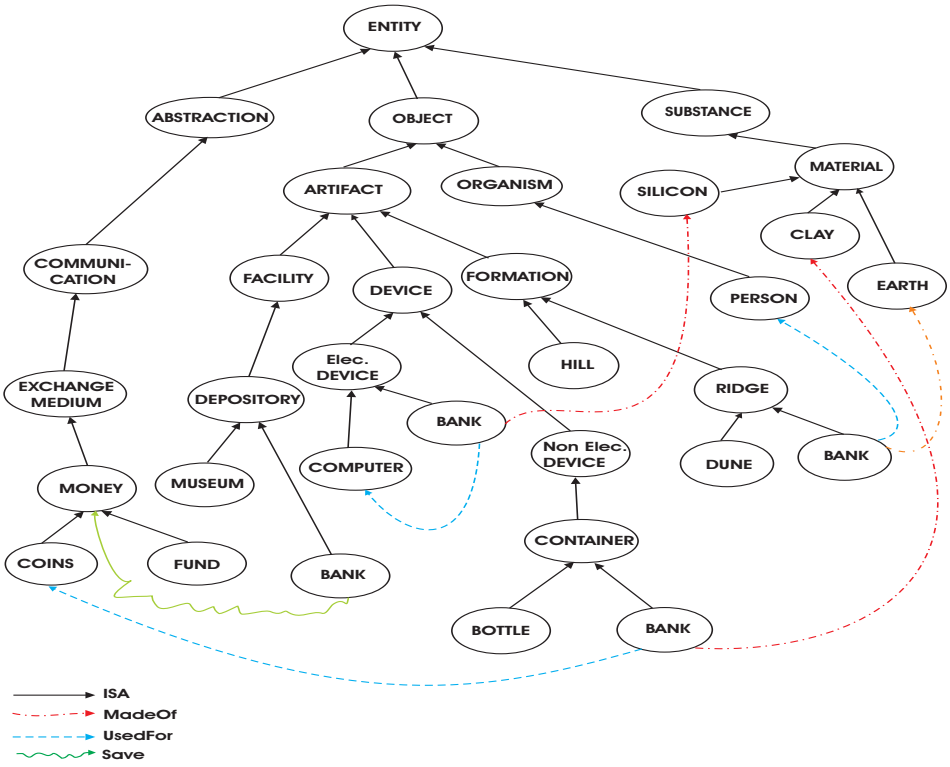


Figure 5.8: Entity Ontology (`En0`)

Both rules state the following conditions on query terms for rewriting Q. They require that (i) Q contains an equal predicate on the attribute A_i and a value T_0, (ii) there are mappings between A_i and R_1 and concepts in ζ, and (iii) there are mappings between T_0 and concept C_0. However, the rules state different conditions on the mapping of relationships from $\Re_0$. On one hand, BASE-SENSITIVITY rule deals with the case in which each relationship β_k in $\Re_0$ is mapped to a pair of attributes in R_1. In this case, there is a set of attributes U in R_1 which participate in the relationship mappings. Each attribute pair is composed from A_i and an attribute from U. In addition, the rule requires that All attributes A_k in U are mapped to concepts c_k in ζ which have sub-concept c_{0k}. That is, each attribute pair correspond to a relationship β_k. Thus, database values T_{0k} which correspond to these concepts are considered for extending Q. The extension is done by substituting the original predicate with conjunctive predicates on attributes A_k and T_{0k}. On the other hand, <1-N>SENSITIVITY rule addresses the case where there exist some relations R_h in Σ_0 that convey the semantic of some domain-specific relationships. Therefore, each relationship β_h in $\Re_0$ is mapped to a foreign key of relation R_h. Moreover, an attribute B_{h2} of R_h is mapped to concept c_h in ζ, ie.,concepts representing values of A_i and B_{h2} are related through β_h. Thus, the values T_h, which correspond to concepts c_{0h} related to C_0, are used for extending Q with conjunctions on B_{h2}. Similar to FEATURE rules SENSITIVITY rules do not include a rule expression for the case in which there are many-to-many correspondences between objects represented in relations (rule of type <N-M>). By ontology definition if a given concept participates in certain domain-

specific relationship then it is related to only one concept through that relationship.

Example 5.8 Given a database ImDB which provides information about items of products. The schema for ImDB schema might contains a relation, called ' Item2', whose schema defines the name of each object, the material it is made of, its use and its price. An extension of ImDB and a description of the relation Item2 are given in Figure 5.9.

Item2(AID, Name, Made, Use, Price)
AID: Item identifier
Name: Item name
Made: Material type
Use: Purpose of use
Price: Item price
PKey(AID)

AID	Name	Made	Use	Price
41	bed	wood	kid	120 $
42	bank	silicon	data	300 $
43	chair	wood	person	150 $
44	flat iron	substance	clothes	60 $
45	chain	gold	women	850 $
46	perfume	roses	women	85 $
47	bank	clay	coins	50 $
48	cage	metal	birds	300 $

Figure 5.9: Item2 relation

In addition, we assume that an ontology called EnO is associated with ImDB. The ontology EnO describes a conceptualization of specific domains of real world entities. A part of this ontology is adopted from the ontology described in [Pea03, Fel98]. Figure 5.8 represents graphically a fragment of EnO.

It is important to note that EnO contains additional domain relationships: "MadeOf", "UseFor" and "Save". The meaning of the "UseFor"-relationship, for example, is that if A (a concept) relates to B (a concept) by this relationship, the objects referred to A are used for purposes given by the objects referred to B. We define the following mappings between EnO and ImDB: mapping between concept ARTIFACT and relation Item2, mapping between concept DEVICE and attribute Name, mapping between concept ENTITY and attribute Use, mapping between sub-concepts of DEVICE and domain values of Name, and mapping between sub-concepts of ENTITY and domain values of Use. We also define mappings between value "bank" and BANK concepts. Furthermore, we also define mappings between relationships "MadeOf" and "UseFor", and attribute pairs ("Name", "Made") and ("Name", "Use"), respectively.

Now, suppose that the user wants to retrieve all tuples from ImDB concerning the container 'bank'. His query can be represented as follows.

$$Q_6 = \{(x_1, x_2, x_3, x_4, x_5) \mid Item2(x_1, x_2, x_3, x_4, x_5) \wedge x_2 = \text{"}bank\text{"}\}.$$

Obviously, the answer from the current ImDB database to the query Q_6 contains the tuples 42 and 47. However, tuple 42 does not meet the intention of the user since it relates to furniture. By using the ontology EnO the system could deduce that "bank" is related to four different contexts: electronic device, non-electronic device, facility, and formation. This is done by retrieving the *immediate uncommon concepts* of BANK

concepts. Therefore, it has to ask again the user for specifying his query providing him the four relevant variants. If the user means a container "bank", the system will be able to specify the concept `BANK` from `EnO` that the related objects are used for keeping coins. Thus, the user query should include terms which are related to the concept `COINS` to assert the intended context of the answer.

Formally, having $T_0 =$ "*bank*", $\beta_3 =$ `"MadeOf"`, and $\beta_4 =$ `"UseFor"` there exist a concept $c_{03}=$ `"CLAY"` which is a sub-concept of $c_3=$ `"SUBSTANCE"` and a concept $c_{04}=$ `"COIN"` which is a sub-concept of $c_4=$ `"ENTITY"`. Thus, based on these informations and the mapping specifications above, the condition of BASE-SENSITIVITY rule is satisfied and hence the rule can be applied to Q_6. The rule extend the query condition by adding equal predicates with conjunctions. These predicates are built over terms $T_{03} =$ "*clay*" and $T_{04} =$ "*coins*". The transformed query is

$$Q_6^{'} = \{(x_1, x_2, x_3, x_4) \mid Item2(x_1, x_2, x_3, x_4) \wedge (x_2 = \text{"}bank\text{"} \wedge x_3 = \text{"}clay\text{"} \wedge x_4 = \text{"}coins\text{"}) \}.$$

Consequently, the answer to this reformulated query will contain only the tuple 47 as expected by the user. □

5.5 The Basic Properties of the Rule Set

In this section we examine the *Termination* and *Confluence* properties of our rewriting system $(\mathcal{T}_e, \mathcal{R}_e)$ as introduced in Section 5.2. We also determine an order for the rules application which does not not produce meaningless results for the transformed queries.

As previously shown we apply the rules to (wff) formulas of query expressions for transforming queries. The transformation process can then be seen as a sequence of rule applications: $F_0 \xrightarrow{*}_{r_i} F_1 \xrightarrow{*}_{r_j} F_2 \ldots$, where r_i, r_j, etc. are rules in $\mathcal{R}_e$, F_0 is a wff of the input query, and F_k are its successors. Since this rewriting is purely syntactical two main problems can arise during the transformation iterations. The first problem is that DRC expressions could be written in several forms which are syntactically dissimilar but logically equivalent. For example, for the formula $\forall x \forall y(p_1(x, y) \Rightarrow \exists\, z\, p_2(x, z))$ it is possible to place all quantifiers in the front of the formula and hence achieving an equivalent formula $\forall x\ \forall y\ \exists\, z\, (\neg\, p_1(x, y) \vee\ p_2(x, z))$. Such problem does not arise with other query languages like Datalog. Our solution is to convert query expression into a *canonical* form. To this end, we use the *prenex disjunctive form* to normalize query formulas.

Definition 5.12 (Prenex Disjunctive Normal Form) *a well formed formula (wff) F is said to be in prenex disjunctive normal form (PDNF) if it has the form*

$$F = \mathcal{K} \bigvee_{i=1}^{s} \Big(\bigwedge_{j=1}^{n_i} p_{ij} \wedge \bigwedge_{j=1}^{m_i} \neg q_{ij} \wedge \psi_i \Big)$$

where p_{ij}'s and q_{ij}'s are predicates (based on the relation schema) that do not contain constants and ψ_i's are build-in predicates, in particular equality ones. $\mathcal{K}$ is a sequence of (existential and/or universal) quantifiers and variables occurring in any predicates.

Example 5.9 Consider three predicates p_1, p_2, and p_3. The following formula is in PDNF

$$\forall x \exists y \exists z \ (p_1(x, y) \ \wedge \ p_2(y, x)) \ \vee \ (p_3(x, z) \ \wedge \ z = 3) \qquad \square$$

We determine a canonical form of a query by converting its formula in PDNF. Therefore, all quantifiers will be placed in front of the formula and predicates will be connected with conjunctions. This form is called *prefix normal form.*

Definition 5.13 (PNF Query Form) *We say that a DRC query Q is in a prefix normal form (PNF) if it is expressed as*

$$Q = \{(x_1, \ldots, x_n) \mid F(x_1, \ldots, x_n)\} \text{ where } F \text{ is a wff written in PDNF}$$

The second problem that may occur during the rewriting is that while matching query patterns with a given rule, syntactical substitution may fail just because (i) the order of predicates in conjunctions (or disjunctions) is not the same as in lhs of the rule, or (ii) the placement of parenthesis '(' and ')' may cause syntactical mismatches. For example, it will be not possible to find a pattern of the form $p_1 \wedge \psi_0$ in the expression $p_1 \wedge p_2 \wedge \psi_0$ unless the order of p_1 and p_2 is permuted. To solve this problem we introduce a set of supplementary rules in order to accommodate the associativity, commutativity, and distributivity properties of logical operators ($\wedge$ and $\vee$) in the rewriting process. Such rules should be taken into account when applying other rules. We call them ACD-Ordering rules .

AC-Ordering Rules:

$$\bar{X} \wedge \bar{Y} \longrightarrow \bar{Y} \wedge \bar{X} \qquad (1)$$
$$\bar{X} \vee \bar{Y} \longrightarrow \bar{Y} \vee \bar{X} \qquad (2)$$
$$\bar{X} \wedge (\bar{Y} \wedge \bar{Z}) \longrightarrow (\bar{X} \wedge \bar{Y}) \wedge \bar{Z} \qquad (3)$$
$$\bar{X} \vee (\bar{Y} \vee \bar{Z}) \longrightarrow (\bar{X} \vee \bar{Y}) \vee \bar{Z} \qquad (4)$$
$$\bar{X} \wedge (\bar{Y} \vee \bar{Z}) \longrightarrow (\bar{X} \wedge \bar{Y}) \vee (\bar{X} \wedge \bar{Z}) \qquad (5)$$
$$\bar{X} \vee (\bar{Y} \wedge \bar{Z}) \longrightarrow (\bar{X} \vee \bar{Y}) \wedge (\bar{X} \vee \bar{Z}) \qquad (6)$$

$\square$

5.5.1 Termination

Given a formula F_0 of a query expression. The rewriting process of F_0 may reach an iteration step after which any further iteration will only result in formulas that are logically equivalent to F_0. We refer to this step number as a *finiteness point.*

Definition 5.14 (Finiteness Point) *Given $F_0 \in \mathcal{T}_e$, and $f \in \mathbb{N}$. We say f is a finiteness point of rewriting F_0 using $r \in \mathcal{R}_e$ iff*

$$F_0 \xrightarrow{f}_r F_l \text{ and } \forall\, i \in \mathbb{N},\, i > f \ \ F_i \equiv F_l$$

We define that a term rewriting system semantically terminates if the rewriting process reach a finiteness point for each term of the system. We call this new property *well-founded termination.*

Definition 5.15 (Well-founded Termination) *We say that a TRS* $(\mathcal{T}, \mathcal{R})$ *is well-founded terminating iff*
$\forall\, t \in \mathcal{T}$, $\forall\, r \in \mathcal{R}\ \exists\, f \in \mathbb{N}$ *such that* f *is a finiteness point of rewriting* t.

Examining the transformation rules of our TRS we deduce that from the syntactical point of view the rewriting does not terminate but from semantic point of view it terminates. This result is proved by the following lemma.

Lemma 5.1 (Rewriting Termination) *The TRS* $(\mathcal{T}_e, \mathcal{R}_e)$ *is well-founded terminating.*

Proof: Let $F_0 \in \mathcal{T}_e$ be a wff formula written in PDNF. To prove that $(\mathcal{T}_e, \mathcal{R}_e)$ is well-founded terminating we show that there exists a finiteness point f of F_0. During the rewriting process the system has to identify subterms of the form $p_i \wedge \psi_i$ for applying rules. We note that there are two cases of rule transformations under substitution $\{\bar{x}_1 \longmapsto x_1, \ldots, \bar{x}_1 \longmapsto x_n\}$: (i) transformations that introduce additional build-in predicates ψ_k (has the form $x_i = T_k$) with disjunctions and (ii) transformations that introduce them with conjunctions. In the first case a given subterm will be transformed into term $p_i \wedge (\psi_i \vee \psi_k)$. Here, for next iterations the system should apply ACD-Ordering rule (5) to convert the predicates in PDNF and hence identifying other subterms. Formally, under the substitution $\{\bar{X} \longmapsto p_i, \bar{Y} \longmapsto \psi_i, \bar{Z} \longmapsto \psi_k\}$ the rewriting has the form $p_i \wedge (\psi_i \vee \psi_k) \longrightarrow (p_i \wedge \psi_i) \vee (p_i \wedge \psi_k)$. In the second case a given subterm will be transformed into the term $p_i \wedge (\psi_i \wedge \psi_k)$. The system here also should apply ACD-Ordering rules (1&3) in order to permute the order of predicates and hence identifying a new subterm for the transformation. Formally, $p_i \wedge (\psi_i \wedge \psi_k) \longrightarrow p_i \wedge (\psi_k \wedge \psi_i) \longrightarrow (p_i \wedge \psi_k) \wedge \psi_i$. We note that the ψ's predicates are defined by terms that are determined using the mapping $\Psi_{\zeta\mathcal{D}}$. Since the number of concepts ζ is finite, the number of predicates ψ_k is also finite (let m be such number). In both transformation cases only redundant predicates are produced after at least m iterations. The new corresponding formulas will be equivalent because these duplicates appear in the same disjunctions (or conjunctions) hence they could be eliminated. Therefore, we conclude that there exists a finiteness point f such that $f \geq m$. In particular, f highly depends on the number of ACD-Ordering rule applications. □

5.5.2 Confluence

In this section we examine the confluence property of $(\mathcal{T}, \mathcal{R})$. We show that the system is confluent for only a subset of rules, namely Synonymy, collection, Part-Whole and Consistency rules.

Let F be a wff involving an atomic formula of the form $R \wedge \psi_0$, where ψ_0 is a predicate built over terms T_0 and x_i. Let also C_0 be a concept in ζ corresponding to T_0. We suppose that F is written as
$F = \bar{\mathcal{F}} \vee (\bar{\mathcal{F}'} \wedge R \wedge \psi_0)$ where $\bar{\mathcal{F}}$ is a formula in PDNF, $\bar{\mathcal{F}'}$ is a conjunction of literals. Let F_1, F_1', F_2, and F_2' be formulas as described in Figure 5.10.

we have to prove that F_2 and F_2' are equivalent for each pair of rules from $(\mathcal{T}_e, \mathcal{R}_e)$. For the sake of clarity, we only deal with the first syntactical expressions of rules. The other expressions could be treated in similar ways.

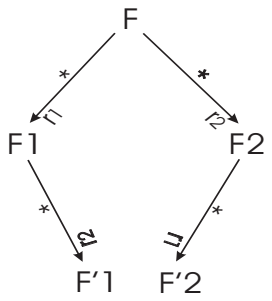


Figure 5.10: Rule Commutation

Lemma 5.2 COLLECTION *and* SYNONYMY *rules commute.*

PROOF: Let r_1 and r_2 be the COLLECTION and the SYNONYMY rules, respectively. The formulas F_1' and F_2' are expressed as
$F_1' = \bar{\mathcal{F}} \vee (\bar{\mathcal{F}'} \wedge R \wedge \psi_0) \vee \bigvee_{k=1}^{m} (\bar{\mathcal{F}'} \wedge R \wedge \psi_k)$ and $F_2' = \bar{\mathcal{F}} \vee (\bar{\mathcal{F}'} \wedge R \wedge \psi_0) \vee \bigvee_{h=1}^{n} (\bar{\mathcal{F}'} \wedge R \wedge \psi_h)$.
Let $L_{F_1'}$ and $L_{F_2'}$ be the sets of distinct ψ_i's predicates in F_1' and F_2', respectively. We show that these sets are identical. Given a predicate $\psi_k \in F_1'$. According to mapping $\Psi_{\mathcal{D}\zeta}$ each term T_k, which builds ψ_k, is mapped into a concept C_k in ζ. Thus, there exists a concept $C_a \in \zeta$ such that C_k satisfies the following formula

$$Isa(x, C_0) \vee Synof(x, C_0) \vee \underbrace{(Isa(C_a, C_0) \wedge Synof(x, C_a))}_{(1')} \qquad (1)$$

On the other hand, each term T_h, which builds ψ_h, is mapped into a concept C_h in ζ. Thus, there exists a concept $C_b \in \zeta$ such that C_h satisfies the following formula

$$Synof(x, C_0) \vee \underbrace{(Synof(C_b, C_0) \wedge Isa(x, C_b))}_{(2')} \qquad (2)$$

Furthermore, based on the following axioms (defined in $\Im$ for the ontology concepts [7])
$\forall\, x\, y\, z\ Isa(x, y) \wedge Synof(y, z) \Rightarrow Isa(x, z)$
$\forall\, x\, y\, z\ Isa(x, z) \wedge Synof(x, y) \Rightarrow Isa(y, z)$
we deduce that formula (1') is equivalent to $Isa(x, C_0)$ and formula (2') is too. This also implies that the formulas (1) and (2) are equivalent. Therefore, any concept C_k satisfies the formula (2) and any concept C_h satisfies the formula (1). We conclude that T_k and T_h build the same predicates of F_1' and F_2', i.e., $L_{F_1'} = L_{F_2'}$ hence the resulting formulas F_1' and F_2' are equivalent. □

Lemma 5.3 COLLECTION *and* SYNONYMY *rules commute with the* PART-WHOLE *rule.*

PROOF: Now, let r_1 and r_2 be the COLLECTION and the PART-WHOLE rules, respectively. The different applications of these rules result in formulas F_1' and F_2' of the form
$F_1' = \bar{\mathcal{F}} \vee (\bar{\mathcal{F}'} \wedge R \wedge \psi_0) \vee \bigvee_{k=1}^{m} (\bar{\mathcal{F}'} \wedge R \wedge \psi_k) \vee \bigvee_{i=0}^{n} \bar{\mathcal{F}'} \wedge p_i$

[7] See Section 3.3

$F'_2 = \bar{\mathcal{F}} \vee (\bar{\mathcal{F}'} \wedge R \wedge \psi_0) \vee \bigvee_{h=1}^{n} (\bar{\mathcal{F}'} \wedge R \wedge \psi_h) \vee (\bar{\mathcal{F}'} \wedge p_0)$
where ψ_i's are build-in predicates and p_i's are atomic formulas. Each p_i has the form of the disjunctive subterm appearing in the PART-WHOLE's rhs.

Let $L_{1F'_1}$ and $L_{1F'_2}$ be the sets of distinct ψ_i's predicates in F'_1 and F'_2, respectively. Furthermore, let $L_{2F'_1}$ and $L_{2F'_2}$ be the sets of p_i's predicates in F'_1 and F'_2, respectively. According to mapping $\Psi_{\mathcal{D}\zeta}$ terms building ψ_i's predicates are mapped into concepts in ζ. Clearly, $L_{1F'_1} = L_{1F'_2}$ since the set of sub-concepts of C_0 is the same for both derivation sequences (ontology is static) hence their corresponding terms build the same predicates. Moreover, each predicate p_i in F'_1 has been built using terms that correspond to parts of C_0. Based on the following axiom (as defined in $\Im$)
$\forall xyz.\ Partof(x, y) \wedge Isa(z, y) \Rightarrow Partof(x, z)$
we deduce that all sub-concepts of C_0 have the same parts as C_0. Therefore, the predicates $p_1, \ldots, p_n$ are duplicates and are equal to p_0. As a result $L_{2F'_1} = L_{2F'_2}$. We conclude then that F'_1 and F'_2 are equivalent.
Similarly, we can prove that SYNONYMY and PART-WHOLE rules commute. We have to reason about synonym concepts of C_0 instead of its sub-concepts. □

Lemma 5.4 SYNONYMY, COLLECTION, *and* PART-WHOLE *rules commute with* CONSISTENCY *rule.*

PROOF: Now, let r_1 and r_2 be the SYNONYMY and the CONSISTENCY rules, respectively. In the first transformations, the rules transform F the formulas into in F_1 and F_2 of the form
$F_1 = \bar{\mathcal{F}} \vee (\bar{\mathcal{F}'} \wedge R \wedge \psi_0 \wedge \bigvee_{k=1}^{m} (R \wedge \psi_k))$
$F_2 = \bar{\mathcal{F}} \vee (\bar{\mathcal{J}} \wedge R \wedge \psi_0)$
where ψ_k's are build-in predicates generated by the rhs of SYNONYMY rule. $\bar{\mathcal{J}}$ is a formula generated by substitutions using the subterm preceding ψ_0 in the rhs of CONSISTENCY rule.
We note that the rules rewrite substitute different subterms in formulas. While CONSISTENCY rule rewrites the predicate $\bar{\mathcal{F}'}$, SYNONYMY rule rewrites predicates $R \wedge \psi_k$. Therefore, in the second transformations the rules transform formulas F_1 and F_2 into
$F'_1 = \bar{\mathcal{F}} \vee ((\bar{\mathcal{J}} \wedge R \wedge \psi_0) \vee (\bar{\mathcal{J}} \wedge R \wedge \psi_1) \vee \ldots \vee (\bar{\mathcal{J}} \wedge R \wedge \psi_k))$
$= \bar{\mathcal{F}} \vee ((\bar{\mathcal{J}} \wedge R \wedge \psi_0) \vee \bigvee_{k=1}^{m} (\bar{\mathcal{J}} \wedge R \wedge \psi_k))$
$F'_2 = \bar{\mathcal{F}} \vee (\bar{\mathcal{J}} \wedge ((R \wedge \psi_0) \bigvee_{k=1}^{m} (R \wedge \psi_k)))$

Thus, we deduce that F'_1 and F'_2 are equivalent. Similarly, we can follow the same reasoning to prove that COLLECTION and PART-WHOLE rules commute with CONSISTENCY rule. □

Theorem 5.1 (Confluent Rewriting) *The TRS* $(\mathcal{T}_e, \mathcal{R}_e)$ *restricted to* SYNONYMY, COLLECTION, PART-WHOLE *and* CONSISTENCY *rules is confluent.*

PROOF: It follows from lemma 5.2, 5.3 and 5.4 that the rules SYNONYMY, COLLECTION, PART-WHOLE and CONSISTENCY commute with each other. Therefore, the TRS $(\mathcal{T}_e, \mathcal{R}_e)$ limited to these rules is confluent. □

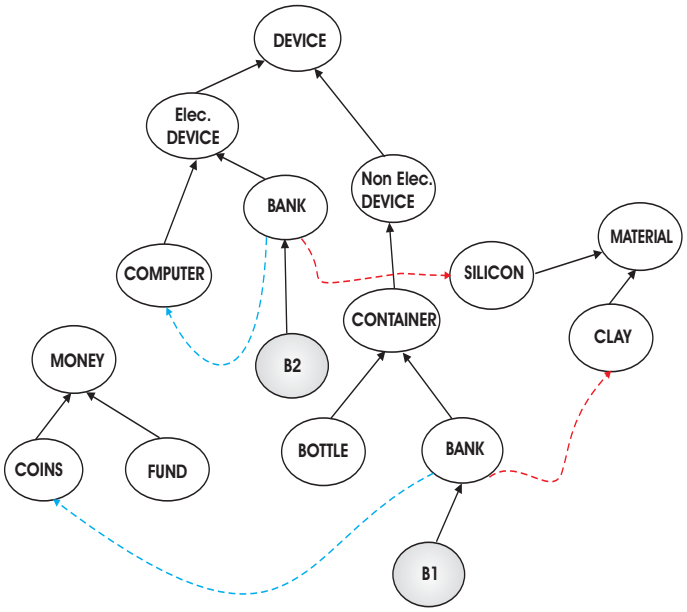


Figure 5.11: An Extension Example of `En0`

5.5.3 The Application Order of Rules

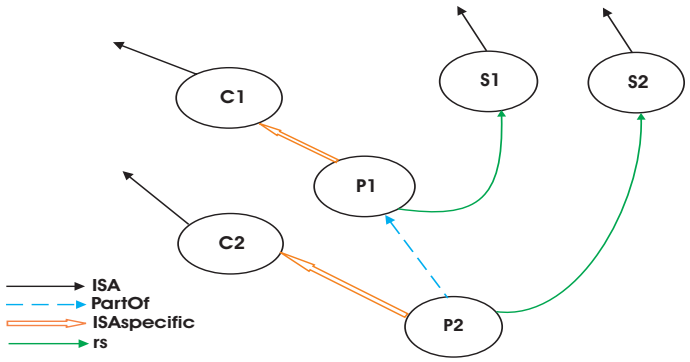
Now, we can divide the $\mathcal{R}_e$ rule set into two subsets. The first subset, $\bar{\mathcal{R}}_e$, consists of rules that commute. They are SYNONYMY, COLLECTION, PART-WHOLE and CONSISTENCY rules. The second subset, $\bar{\mathcal{R}'}_e$, consists of rules that do not commute with any other rule. They are SUPPORT, FEATURE and SENSITIVITY rules. One of the problem that may arise by arbitrary applying all the rules to a query expression is that the answer of the query may contain false positives. This is caused by the application of rules that do not commute. To solve this problem we propose in this section an application order of the $\mathcal{R}_e$ rules and illustrate why it is suitable. Notice that our choice of the order is independent of how it affects the performance of query execution. We order the rules such that we first apply $\bar{\mathcal{R}'}_e$ rules and then $\bar{\mathcal{R}}_e$ rules as follows.

Definition 5.16 (Proposed Application Order) *Given a query Q and a TRS $(\mathcal{T}_e, \mathcal{R}_e)$. We define the order of $\mathcal{R}_e$ applied to Q as*

$$Q \xrightarrow{*}_{ST} Q_1 \xrightarrow{*}_{SP} Q_2 \xrightarrow{*}_{FT} Q_3 \xrightarrow{*}_{r_1} \ldots \xrightarrow{*}_{r_4} Q_7$$

where $r_i \in \bar{\mathcal{R}'}_e$ and ST, SP, and FT denote SENSITIVITY, SUPPORT, *and* FEATURE *rules, respectively.*

There are two main reasons why we should apply a SENSITIVITY rule before other rules. First, it is necessary to specify the context of the query and hence eliminate irrelevant tuples as early as possible. This task is supported by such rule. Second, a SENSITIVITY rule does not commute with any other rules and any preceding application of another rule would not correctly transform the query within the specified context as illustrated in the following example.



Test

ID	A1	A2	A3
1	v1	vp1	vs1
2	v2	vp2	vs2
3	v3	vc1	vs1
4	v4	vc2	vs2

Figure 5.12: An illustrative Example

Example 5.10 Supposing that the `BANK` concepts in `EnO` have sub-concepts `B1` and `B2`, as depicted in Figure 5.11, and relation Item2 contains two additional tuples (v_1, v_{B1}, "clay", "coins", v_5) and (w_1, w_{B2}, "silicon", "data", w_5). We assume that `BANK` concepts are mapped into v_{B1} and w_{B2}. Now, we examine the following rewritings:
$Q \xrightarrow{*}_{ST} Q_1 \xrightarrow{*}_{CR} Q'_1$ and $Q \xrightarrow{*}_{CR} Q_2 \xrightarrow{*}_{ST} Q'_2$
where ST and CR refer to Sensitivity and Collection rules, respectively. Let Q be query Q_6 of the example 5.8, we obtain then

$Q'_1 = \{(x_1, x_2, x_3, x_4) \mid Item2(x_1, x_2, x_3, x_4) \wedge [(x_2 = "bank" \wedge x_3 = "clay" \wedge x_4 = "coins") \vee (x_2 = v_{B1} \wedge x_3 = "clay" \wedge x_4 = "coins")]\}$.

$Q'_2 = \{(x_1, x_2, x_3, x_4) \mid Item2(x_1, x_2, x_3, x_4) \wedge [(x_2 = "bank" \wedge x_3 = "clay" \wedge x_4 = "coins") \vee x_2 = v_{B1} \vee x_2 = w_{B2}]\}$.

The identifiers of tuples returned by queries Q'_1 and Q'_2 are $\{47, v_1\}$ and $\{47, v_1, w_1\}$, respectively. Therefore, Q'_1 and Q'_2 are not equivalent. We notice that the second rewriting result in a false positive (tuple identified by w_1). This is due to the fact that the application of Sensitivity rule to Q_2 does not introduce additional constraints to all conjunctions in the query expression. □

Now, let us have these rewritings: $Q \xrightarrow{*}_{SP} Q_1 \xrightarrow{*}_{FT} Q'_1$ and $Q \xrightarrow{*}_{FT} Q_2 \xrightarrow{*}_{SP} Q'_2$ where SP and FT refer to Support and Feature rules, respectively. The main reason why we should apply Support rules before Feature rules for transforming queries is that these rules do not commute and the results of transformed query Q'_1 is contained in Q'_2. That is, the answer of Q'_2 would contain more relevant tuples than the answer of Q'_1. We illustrate this fact by the following example.

Example 5.11 Supposing that we have a set of concepts $\{C_1, C_2, P_1, P_1, P_2, S_1, S_2\}$ which appear in an ontology as shown in Figure 5.12. In addition, suppose we have a re-

lation `Test` which is associated to that ontology. The association define concept-value mappings between concepts C_1, C_2, P_1, P_2 and values vc_1, vc_2, vp_1, vp_2, respectively and relationship-attribute pair between relationships `PartOf`, `rs` and attribute pairs (A_1, A_2), (A_2, A_3), respectively. Assuming we submit this query

$Q = \{x_1 \mid \exists\, x_2\, x_3\, x_4\ Test(x_1, x_2, x_3, x_4) \wedge x_3 = \text{"}vp_2\text{"}\}$.

While the first rewriting result in

$$Q'_1 = \{x_1 \mid \exists\, x_2\, x_3\, x_4\ Test(x_1, x_2, x_3, x_4) \wedge [(x_3 = \text{"}vp_2\text{"} \vee x_3 = \text{"}vp_1\text{"}) \vee (x_3 = \text{"}vc_1\text{"} \wedge x_4 = \text{"}vs_1\text{"}) \vee (x_3 = \text{"}vc_2\text{"} \wedge x_4 = \text{"}vs_2\text{"})]\}$$

the second rewriting result in

$$Q'_2 = \{x_1 \mid \exists\, x_2\, x_3\, x_4\ Test(x_1, x_2, x_3, x_4) \wedge [(x_3 = \text{"}vp_2\text{"} \vee x_3 = \text{"}vp_1\text{"}) \vee (x_3 = \text{"}vc_2\text{"} \wedge x_4 = \text{"}vs_2\text{"})]\}.$$

Thus, the answers of Q'_1 and Q'_2 are $\{1, 2, 4\}$ and $\{1, 2, 3, 4\}$, respectively. We can deduce that Q'_1 is contained in Q'_2. □

Finally, we argument the application of $\bar{\mathcal{R}}_e$ rules after $\bar{\mathcal{R}'}_e$ rules by the fact that any two rules from the distinct sets do not commute and any rule of $\bar{\mathcal{R}'}_e$ following a rule of $\bar{\mathcal{R}}_e$ will lead to meaningless results as illustrated in Example 5.10.

5.6 Summary

Rules		**Mappings**						
Rule Set	**Rule Name**	$\Psi_{\zeta\mathcal{D}}$	$\Psi_{\Re\mathcal{E}}$	$\Psi_{\zeta\mathcal{U}}$	$\Psi_{\zeta\Sigma}$	$\Psi_{\Re\mathcal{P}}$	$\Psi_{\Re\Sigma}$	$\Psi_{\Re\mathcal{F}}$
Synonym		✓		✓	✓			
Collection		✓		✓	✓			
Part Whole	Base Part Whole	✓		✓	✓	✓		
	<1-n>Part Whole	✓		✓	✓			✓
	<n-m>Part Whole	✓		✓	✓		✓	
Support	Base-Support	✓		✓	✓	✓		
	<1-n>Support	✓		✓	✓			✓
	<n-m>Support	✓		✓	✓		✓	
Feature	Base-Feature	✓		✓	✓	✓		
	<1-n>Feature	✓		✓	✓			✓
Consistency	<1-n>Consistency		✓		✓			✓
	<n-m>Consistency		✓		✓		✓	
Sensitivity	Base-Sensitivity	✓		✓	✓	✓		
	<1-n>Sensitivity	✓		✓	✓			✓

Table 5.2: Use of Ontology-Database Mappings by Transformation Rules

We presented in this chapter a new approach to query processing using additional semantic knowledge. Our goal is to give DBMSs the ability to deal with user queries both at the semantic and the syntactic levels. Therefore, DBMSs will be able to generate

answers which meet the intention of the user. These answers are partially independent from the query syntax. We claim that ontologies can provide a good support to achieve this goal. We illustrated how an ontology could be effectively exploited to capture semantics of a database for transforming user queries. As a result, a transformed query could provide more meaningful answer than the original one. We presented two categories of such rules: rules that might extend the results of the original query and others that might reduce them. We outlined the basic features of each rule and illustrated their effectiveness and their usefulness by means of some practical applications.

The transformation rules use mapping information which are required to integrate ontology semantics in the query. To reformulate a given query mappings are established based on database components and user's terms building the query expression. In Table 5.6 we summarize the different kinds of mappings used by the rules. We recall that $\Psi_{\zeta\mathcal{D}}$, $\Psi_{\Re\mathcal{E}}$, $\Psi_{\zeta\mathcal{U}}$, $\Psi_{\zeta\Sigma}$, $\Psi_{\Re\mathcal{P}}$, $\Psi_{\Re\mathcal{F}}$, and $\Psi_{\Re\Sigma}$ define mappings between concepts and database values, relationships and database extensions, concepts and attributes, concepts and relations, relationships and attribute pairs, relationships and key attributes, and relationships and relations, respectively.

Finally, we made use of the theory of term rewriting systems to formalize our query transformation and to determine the main properties for applying the transformation rules. We defined a rewriting system for queries and prove that it terminates and is confluent for most of rules. In addition, we proposed an order for applying all the rules and prove that it leads to a meaningful answer for any query.

Chapter 6

ODBT-Prototype: From Concepts to Realization

The concepts developed in this thesis are not just of theoretical interest, but of practical importance as well. For this reason, we implemented a prototype called "Ontology Database Transformer" (ODBT). This chapter briefly discusses implementation aspects for the prototype. We first investigate how the existing language and tools can be used for implementing ontologies and develop an approach for representing the mappings. Then, we illustrate the main modules of the prototype and their functionalities. Finally, we evaluate the capabilities of the prototype using a real world application. The capabilities are evaluated in terms of our problem definition. This means that the prototype is not designed and implemented with concerns such as efficiency, flexibility, or maintainability. However, we concentrated on the quality of the query answer resulting from the semantic transformation.

6.1 Ontology Languages and Storage Tools

This subsection briefly present the languages and associated tools which are appropriate for implementing ontologies in our system. The comparison and evaluation of different ontology languages and tools is out of the scope of this thesis. Good surveys of these topics can be found in [PC02, CPCF04, MKC+02, GGCD02].

In recent years, several ontology languages have been developed. However, the languages that are developed for Web applications become the most prominent and are undergoing standardization. While some of these languages are based on XML syntax, such as OXL (Ontology Exchange Language) [KCT06] and SHOE (Simple HTML Ontology Extensions) [HHL03], others are based on RDF (Resource Description Framework), such as RDF/S [KC04], OIL [FHvH+00], DAML+OIL [Hor02], and OWL [MH06]. We opted for standard languages that conform to our ontology design and that allow us to reason with ontological data.

6.1.1 Using RDF for Modelling Ontologies

RDF is developed by W3C (World Wide Web Consortium) for a standard representation of meta-data [KC04]. The objective of RDF is to describe Web resources in an interoperable and human readable way without making assumption about the application domain or the structure of the described resource. RDF is a simple, yet a powerful data model [Ogb00]. It consists of three building blocks:

1. *Resources* denoted by unique identifiers (URIs) for representing real world objects or abstract objects as well as statements that describe binary relations between these objects.

2. *Properties* which specify aspects, characteristics, or attributes for describing resources.

3. *Statements* which have a triple-form *(Subject, Predicate, Object)*, where all three components are resources of an RDF model. They can be seen as a set of logical axioms representing facts in the real world. Therefore, an RDF model can be interpreted by the conjunction (i.e., logical AND) of all the statements it contains. A statement corresponds to a simple sentence in natural language, e.g., 'Gabi loves Hans'. Here, 'Gabi' corresponds to the subject, 'loves' corresponds to the predicate, and 'Hans' corresponds to the object [Ogb00].

Use of RDF. A RDF model can be seen as a directed labelled graph. The nodes of this graph are RDF resources and the edges are RDF properties. Therefore, we can simply map an ontology graph G to an RDF model: Each edge in G could be encoded by an RDF statement, where the edge-nodes (concepts) correspond to the subject and the object, and the edge-label (relationship) correspond to the predicate. In RDF ontology concepts are written with a prefix 'rdf'. In comparison with other representation languages RDF is simple. However it does not provide a mechanism for defining how RDF resources and properties could be combined in a description. In companion with a schema, called RDFS, RDF can be used to define this description. RDF and RDFS are referred to as RDF/S. RDFS extends RDF by offering additional primitives for defining RDF concepts. That is, they can be viewed as a meta-data about RDF elements. Much of these primitives are inspired from object-oriented programming. RDFS offers classes for both resources and properties [BG06]. Defining objects in classes is very attractive for describing resources because it allow us to derive more special resource classes from more general ones. Some important RDFS classes are:

- *rdfs:Resource*: It is the basic class for RDF resources. Every resource is an *rdfs:Resource*.

- *rdfs:class*: It is a subclass of rdfs:Resource. Every resource belongs to the class *rdfs:Resource* and all classes used in RDFS are subclasses of resources.

- *rdf:type*: It is used to specify that a resource is of a certain type.

- *rdfs:subClassOf*: It is an important property that applies to RDF resources which are of *rdf:type* class. It is used to specify subclasses of another class.
- *rdf:Property*: It defines all properties in RDF. Properties can be attributes of a resource or relationships between resources.
- *rdfs:SubPropertyOf*: It is used to specify sub-properties of another property. That is, if P is a property for two given resources and P' is a sub-property of P then P' is a property for that sources.
- *rdf:statement*: It is a subclass of the class *rdfs:Resource* and describes the triple statements.

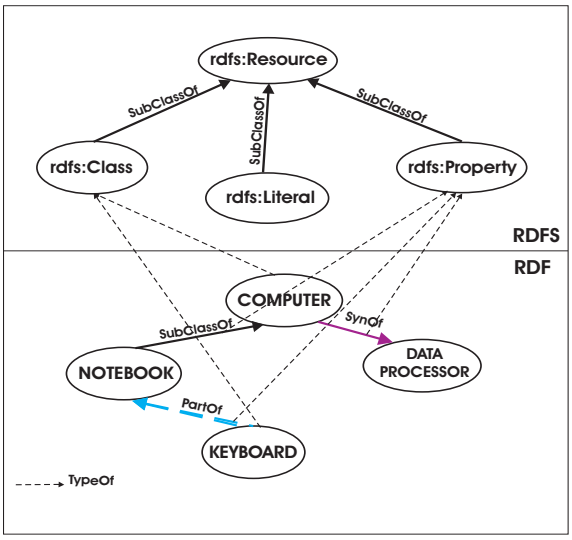


Figure 6.1: Fragment of `PO`-ontology in RDF/S

Given an ontology $O = \{G, \zeta, \Re, \Im\}$, we can model the ontology using RDF/S as follows. First, all concepts and relationships are expressed as *rdfs:Resource*. Notice that in RDF/S ontology elements are written with a prefix 'rdfs'. Second, relationships of $\Re$ are built in terms of *rdf:statement* where *rdfs:Resource* having a type *rdf:predicate* corresponds to $\Re$. Furthermore, concepts of ζ are defined by *rdfs:class* and can occur in relationships as *rdf:subject* or *rdf:object*. It is important to express in RDF/S the properties of relationships such as transitivity for `ISA`, `SynOf`, and `PartOf` as defined in $\Im$. This will be useful for building inferences about concepts. However, RDF/S can express transitivity only for `ISA` by means of *rdfs:subClassOf*. For `SynOf` and `PartOf` relationhships other modelling primitives will be needed. Figure 6.1 presents visually the RDF/S of a part of the ontology `PO`. The upper portion of the figure refers to the corresponding RDF schema while the lower portion refers to the corresponding RDF model.

```
01 <?xml version="1.0"?>
02 <!DOCTYPE rdf:RDF [<!ENTITY xsd
"http://www.w3.org/2001/XMLSchema#">]>
03 <rdf:RDF xml:lang="en"
04 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
05 xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">
06 <rdfs:Class rdf:ID="Computer">
07 <rdfs:comment>device that computes</rdfs:comment>
08 </rdfs:Class>
09 <rdfs:Class rdf:ID="DataProcessor">
10 <rdfs:comment>device that computes</rdfs:comment>
11 </rdfs:Class>
12 <rdfs:Class rdf:ID="Notebook">
13 <rdfs:comment>light, portable computer</rdfs:comment>
14 <rdfs:subClassOf rdf:resource="Computer"/>
15 </rdfs:Class>
16 <rdfs:Class rdf:ID="Keyboard">
17 <rdfs:comment>set of keys</rdfs:comment>
18 </rdfs:Class>
19 <rdf:Property rdf:ID="SynOf">
20 <rdfs:comment>having the same meaning</rdfs:comment>
21 </rdfs:Property>
22 <rdf:Property rdf:ID="PartOf">
23 <rdfs:comment>set of</rdfs:comment>
24 </rdfs:Property>
25 <rdf:Statement>
26 <rdf:subject rdf:resource="#DataProcessor"/>
27 <rdf:predicate rdf:resource="#SynOf"/>
28 <rdf:object rdf:resource="#Computer"/>
29 </rdf:Statement>
30 <rdf:Statement>
31 <rdf:subject rdf:resource="#Keyboard"/>
32 <rdf:predicate rdf:resource="#PartOf"/>
33 <rdf:object rdf:resource="#Notebook"/>
34 </rdf:Statement>
35 </rdf:RDF>
```

Figure 6.2: RDF/S syntax of example in Fig. 6.1

RDF/S-XML syntax. As previously seen, RDF/S defines a syntax independent model for representing resources. However, we need a syntax representing this model to manage and exchange RDF data in a machine-readable way. XML language is a good candidate for providing such syntax as specified in [KC04]. Following this specification we illustrate this syntax by means of an example as shown in Figure 6.2. In lines 04-05 namespaces of RDF and RDFS must be defined such that we can use prefixes '*rdf*' and '*rdfs*' for RDF and RDFS concept definitions within the rest of document. In lines 6, 9, 12, and 16 a class for each concept `COMPUTER`, `DATA PROCESSOR`, `NOTEBOOK`, and `KEYBOARD` is defined. Each class has a property 'comment' which is modelled by *rdfs:comment*. In particular, the class `NOTEBOOK` has an additional property that is it is a subclass of `COMPUTER` modelled by *rdfs:subClassOf* primitive. In lines 19 and 22 `ISA` and `PartOf` relationships are defined as resources. In lines 25-29 and in lines 30-34 statements are used to define the synonym relationship between `DATA PROCESSOR` and `COMPUTER`, and the part-whole relationship between `NOTEBOOK` and `KEYBOARD`.

We should note that this part of ontology is not completely mapped into RDF/S since semantics of `PartOf` and `SynOf` are not expressed and hence they could be not used for inference computations. Thus, we conclude that RDF/S is not sufficient for implementing ontologies of our prototype. We need more expressive modelling primitives for representing these semantics.

6.1.2 Using OWL for Modelling Ontologies

Another alternative for modelling ontologies is the use of OWL language. OWL is an extension of RDF/S and hence supports RDF modelling concepts as described in the last section. It is a revision of the DAML+OIL language and has been recommended by W3C as a language for designing ontologies in the Web. OWL has three sub-languages: *OWL-Lite*, *OWL-DL*, and *OWL-Full*, each with different expressive power [MH06].

OWL represents concepts as classes which are referred to as *owl:Class*. OWL offers more facilities for expressing semantics than RDF and RDFS. It adds more modelling features for describing properties and classes, among others, are the following:

- Relations between classes: e.g., *owl:disjointWith* states that two classes have distinct instances.
- Boolean combinations of classes: e.g., *owl:intersectionOf*, *owl:unionOf*, and *owl: complementOf* allow combinations of classes using AND, OR, and NOT operators.
- Enumerated classes: e.g., *owl:oneOf* states that the members of the class are exactly the set of enumerated instances in the class.
- Quantification: e.g., *owl:allValuesFrom* and *owl:someValuesFrom* represent the `all` quantifier and `exist` quantifier respectively.
- Additional characteristics: e.g., equality, inequality, transitivity, and symmetry between properties. Some of classes representing equality and inequality are: *owl:equivalentClass*, *owl:differentFrom*, *owl:sameAs*. Classes for transitivity and symmetry are *owl:Transitive Property* and *owl:SymmetricProperty*, respectively.

By using OWL we are now able to express additional semantic properties of relationships such as equivalence, symmetry, and transitivity. Furthermore, we can represent the `SynOf`-relationship by the class *owl:equivalentClass*. Unfortunately, there are no specific constructs in OWL for representing the `PartOf`-relationship. OWL does not have sufficient facilities to express much of what one may want to represent about `PartOf` properties. This problem has been addressed in [MH06] where some solutions have been proposed "*OWL does not provide any build-in primitives for part-whole relations, but contains sufficient expressive power to capture most, but not all, of the common cases*" [MH06]. The `PartOf`-relationship is a partial order, i.e., it is transitive, anti-symmetric, and reflexive. However, OWL can only express transitivity property. Other properties can be expressed by means of some OWL build-in primitives as described in [MH06].

An XML syntax for OWL has been developed based on the RDF/S syntax described in Section 6.1.1. It adopts concepts from RDF/S and uses the same notations to represent them. Figure 6.3 shows how the fragment of `PO` described in Figure 6.2 can be represented in the syntax of OWL. In line 11, equivalence between concepts `COMPUTER` and `NOTEBOOK` are expressed using the class *owl:equivalentClass*. The inverse relationship of `PartOf`, called `HasPart`, can be defined using the construct *owl:inverseOf* as shown in line 31. In addition, transitivity of `PartOf` and `HasPart` relationships are

```
01 <?xml version="1.0"?>
02 <rdf:RDF
03 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
04 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
05 xmlns:owl="http://www.w3.org/2002/07/owl#"
06 <owl:Ontology rdf:about=""/>
06 <owl:Class rdf:ID="Computer">
07 <rdfs:comment>device that computes</rdfs:comment>
08 </owl:Class>
09 <owl:Class rdf:ID="DataProcessor">
10 <rdfs:comment>device that computes</rdfs:comment>
11 <owl:equivalentClass>Computer</owl:equivalentClass>
12 </owl:Class>
13 <owl:Class rdf:ID="Notebook">
14 <rdfs:comment>light, portable computer</rdfs:comment>
15 <rdfs:subClassOf rdf:resource="Computer"/>
16 </owl:Class>
17 <owl:Class rdf:ID="Keyboard">
18 <rdfs:comment>set of keys</rdfs:comment>
19 <rdfs:subClassOf>
20 <owl:Restriction>
21 <owl:onProperty>
22 <owl:ObjectProperty rdf:ID="is_part_of"/>
23 </owl:onProperty>
24 <owl:someValuesFrom>
25 <owl:Class rdf:ID="Notebook"/>
26 </owl:someValuesFrom>
27 </owl:Restriction>
28 </rdfs:subClassOf>
29 </owl:Class>
30 <owl:TransitiveProperty rdf:about="#is_part_of">
31 <owl:inverseOf>
32 <owl:TransitiveProperty rdf:about="#has_part"/>
33 </owl:inverseOf>
34 <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
35 </owl:TransitiveProperty>
36 <owl:TransitiveProperty rdf:about="#has_part">
37 <owl:inverseOf>
38 <owl:TransitiveProperty rdf:about="#is_part_of"/>
39 </owl:inverseOf>
40 <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
41 </owl:TransitiveProperty>
42 </rdf:RDF>
```

Figure 6.3: OWL-XML syntax of example in Fig. 6.1

explicitly defined in lines 30 and 36, and the inverse relation of each other is defined in lines 31-33 and lines 37-39. We also specify in lines 22-23 that concept `KEYBOARD` is part of `NOTEBOOK`. We restrict properties `KEYBOARD` to only a subset from the properties of `NOTEBOOK` using the construct *owl:someValuesFrom*, as indicated in lines 24-26.

It has been shown that OWL is an appropriate language for our prototype since it satisfies our requirements for modelling ontologies as defined in Chapter 3. In comparison with RDF/S OWL offers more build-in primitives that are useful for specifying semantics of the ontology relationships. Currently, there exist many ontologies represented by this language and many tools for supporting their implementation are emerging. By the recommendations of W3C OWL intend to be a standard technology for future ontology based applications on the Web.

6.1.3 Jena and Seasme

There are many tools for storing and querying ontologies [MKA+02]. Given the experimental nature of our prototype, requirements like response time, reliability (typically

decreasing with increasing role of inference), and quality of editing interface do not play a very important role, and since we deal with a single ontology fully under our control, there is no need for dynamic integration mechanisms. However, we need tools that obey the following requirements:

- **Storage options:** Ontologies must be implemented either in relational databases (persistent storage) or in internal memory (in-memory storage). In our setting for the storage, we opted for an RDBMS back-end.
- **RDF format:** Tools should be able to access RDF data in XML format, whereby appropriate methods for creating, reading, updating and querying data from ontology are available.
- **Ontology language:** Tools should enable to implement OWL language primitives and its inference mechanisms.
- **Inference support:** It is necessary that the tools permit arbitrary deduction rules for inferring new statements. We need two kinds of support for the inferring process: (i) support for integrating the rules with the query language of DBMS in seamless way, and (ii) support for adding new rules to the existing inferring engine in an easy way. The second one allows us to define our inference rules.
- **API support:** APIs provides functions for querying and updating ontologies and can be used for interfacing with clients. Hence, they offer developers the ability to deploy their applications on the top of these tools. For our query transformation, we need to access the asserted statements as well as the inferred statements from an ontology. That is, we need to access two graphs: the *initial graph* of the ontology and the *inferred graph* created by inference rules of the language.

In the following, we briefly describe two well-known tools for storage and retrieval RDF data, *Jena* and *Sesame*, and analyze their applicability to our prototype with respect to the requirements above.

Jena. Jena [McB02] is an open-source project developed by Hewlett-Packard [1]. One of the main powerful feature of Jena is that its API is flexible and expressive for managing RDF graphs. Users have the choice to store RDF graphs either in memory or in the database. Jena2 [2] includes an RDF/XML Parser, an I/O module, and a query system based on RDQL (RDF Data Query Language) language [HBEV04]. The internal structure of Jena is based on a simple abstraction of RDF graphs as triple-based structure. The database subsystem of Jena allows a persistent implementation for RDF graphs using a JDBC interface. It supports different database managements systems such as MySQl, Oracle, and Postgresql.

Jena has a modular architecture endowed with a set of reasoning engines called *Reasoner* [3] which are developed for RDF/S, OWL, or DAML+OIL languages. The

[1] URL: http://jena.sourceforge.net

[2] The second generation of Jena

[3] A module for implementing inference

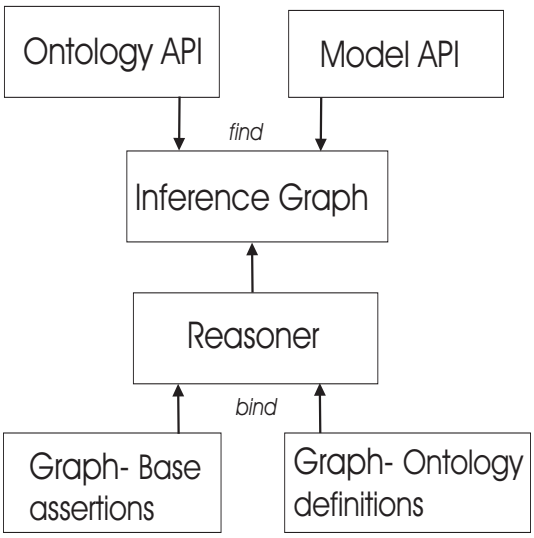


Figure 6.4: Inference System in Jena

reasoner creates inferred graphs using the initial ontology graph (basic graph) and a set of inference rules. In addition, it allows users to define their own inference rules. The inferred graphs are supported by the same access and query interfaces as the basic graph. However, Jena provides additional methods to check the consistency of the resulting inferred statements in these graphs. As shown in Figure 6.4, inferred graphs are layered on top of the basic graph. The reasoner is able to assign instance data (in case where ontologies contains instances) to concepts of the graph. This is established by the method *bind* which has the ontology graph and instances as input and generate virtual triples as output. The method *find* is used to retrieve not only those statements that were present in the original ontology but also additional statements derived from inferred graph. Another task of the reasoner is the handling of transitive relationships [WSKR03]. The corresponding rules used for this task are defined in the ontology API. For instance, the following rule expresses the transitivity of `ISA`-relationship.

```
[rule 2: (?a rdfs:subClassOf ?b)
         (?b rdfs:subClassOf ?c)
          ⟶ (?a rdfs:subClassOf ?c)]
```

Figure 6.5: Jena Rule for the Transitivity of `ISA`

Jena provides support for storing inferred graphs only in a compact in-memory form. It uses efficient caching mechanisms that improve performance of querying the graph. However, we reported in our implementation performance penalties when Jena executes the first queries on the inferred graph since the graph does not yet exist. To cope with this problem some methods have been proposed for future versions. One of these methods is to store parts of the inferred graph (e.g., the transitive closure of `ISA`) in the persistent storage.

The language used in Jena to query RDF data is RDQL. Its main operation is to find pattern in graphs, i.e., return all statements satisfying a pattern of the form (S,P,O). An RDQL query is converted into a pipeline of pattern operations connected by join variables to specify joins. Jena tries to push joins into the DBMS for evaluation. The goal then is to convert multiple patterns into a single query to be evaluated efficiently.

```
<rule name="odbt6">
        <premise>
                <subject var="xxx"/>
                <predicate uri="&hpa;hasPart"/>
                <object var="yyy"/>
        </premise>
        <consequent>
                <subject var="yyy"/>
                <predicate uri="&pof;partOf"/>
                <object var="xxx"/>
        </consequent>
        <triggers_rule>
                <rule name="odbt7" />
        </triggers_rule>
</rule>
```

Figure 6.6: Sesame Rule for the Inverse of `PartOf`

Sesame. Sesame [BKvH02], developed by Aidministrator Nederland [4], is an open-source Java framework for storage and quering RDF data and schema. Sesame architecture consists of three modules: a *query engine*, an *administration module*, and an *export module*. The query engine enables to parse a query, build the query tree to optimize it, and finally evaluate it in a streaming fashion. The administration module enables to insert and delete RDF data and schema, and check the consistency of newly inferred statements. The export module extract data and schema information and export them in XML/RDF format [Gro03].

Sesame supports several storage options. It allows persistent storage in form of databases or files as well as in-memory storage. To this end, it uses a stack of `SAILs` (Storage and Inference Layers), which transparently ensures access to specific implementations of repository. The underlying repositories can be based on a relational (or object-oriented) DBMS such as MySQL, PostgreSQL, Oracle, and SQLServer. RDF files are cached in memory, in particular, for little amount of data. Furthermore, sesame offers additional storage option whereby data are stored in files based B-Tree techniques. It also offers access to distributed RDF data based on network services or existing RDF repositories. Sesame provides a good performance for storing data in main memory unless the number of triple is more than one million. However, it needs more memory space and more time for the initial phase.

[4]URL: http://sesame.aidministrator.nl

To facilitate querying, Sesame implements three query languages: RQL, RDQL, and SeRQL [SSKL04]. Therefore, it supports the basic inferencing needed for RDF schema build-in primitives such as subClassOf- and subPropertyOf properties. Inference is done according to rules and axioms defined by W3C for RDF. For this purpose, Sesame offers an extra module which implements a forward chaining algorithm. It calculates and stores the transition closure during the transaction that adds data to the data repository. In addition, Inferencing can be done by defining application-specific rules. These rules can be expressed in OWL and stored in a separate XML-file. In our system, for example, it is possible to define a rule stating that property `HasPart` is inverse to property `PartOf`. We recall that this rule is given by the axiom

We notice that the syntax of the rule adheres to the RDF-Graph through naming XML-tags 'subject', 'predicate', and 'object'. We also notice that a premise is stated for each statement which implies a consequence whenever it is evaluated to True. As opposed to Jena, Sesame enables to define additional triggers. These triggers may be fired during the application of a given rule in order to execute other rules. The goal of using triggers is to prevent the applications of useless rules. For instance, the rule 'odbt7' presented in Figure 6.6 is triggered by the rule 'odbt6'.

The programming interface of ODBT, as described in Section 6.3, enables access to ontology by means of the previous tools. The implemented methods for query transformations must be able to access ontology graph as well as inferred graph directly through each tool. Table 6.1 summarizes all requirements that must be accomplished to implement ODBT ontologies and those which are satisfied by Jena and Sesame. As we can see, Sesame has more functionalities than Jena, e.g., Jena does not offer persistent storage for inferred graphs.

	Jena	**Sesame**
Impl. Language	Java	Java
DBMSs	Mysql, PostgreSql, Oracle, BerkeleyDB	Mysql, PostgreSql, Oracle
Query Language	RDQL	RQL, SeRQL, RDQL
RDF/S	Yes	Yes
RDF-XML	Yes	Yes
OWL	Yes	Yes
OWL Inference	Yes	Yes
User-defined Inference	Yes	Yes
Persistent Inference Graph	No	Yes
Availability	Open-source	Open-source

Table 6.1: Overview of Ontology Tools

6.2 Mapping Representation

The literature references few approaches to the problem of representing mappings between ontologies and databases [Biz03, BCP04]. In addition, these approaches are not appropriate for realizing our ontology-based transformation. They are not able to express mappings between ontology elements and database extensions as illustrated in Chapter 4. In this section, we briefly describe our approach for representing mappings between relational databases and ontologies implemented in RDF/S or OWL. We propose a language that intends to extend and enhance the mapping capabilities of two existing languages, D2R [Biz03] and R2O [BCP04]. Our language is fully declarative and uses XML syntax to reference ontologies and database components. Appendix B.2 presents the XML data-type definitions for mappings between ontologies and databases described in previous chapters. To illustrate the language we describe how to express $\Psi_{\Re\mathcal{E}}$ mappings linking ontology relationships with relation extensions. An element of such mappings consists of a relationship β and two extension-fields $E_{projSub}$ and $E_{projObj}$. That is, all instances of $E_{projSub}$ (referred to as Subject) are mapped to all instances of $E_{projObj}$ (referred to as Object) through the relationship β. We use the following example to illustrate how we can express this mapping.

```
<DbRelationOntRelationshipMapping>
    <DbRelation name="COMPONENT" />
    <OntRelationship uri="http://localhost/ontologies/partWhole/#partOf" />
    <ProjectionFrom>                                          Foreign Key 1
        <DbAttribute name="MID" />                            Extension-field 1   Subject
        <DbRelation name="PRODUCT">
            <DbAttribute name="NAME" />
        </DbRelation>
    </ProjectionFrom>
    <ProjectionTo>                                            Foreign Key 2
        <DbAttribute name="SID" />                            Extension-field 2   Object
        <DbRelation name="PRODUCT">
            <DbAttribute name="NAME" />
        </DbRelation>
    </ProjectionTo>
</DbRelationOntRelationshipMapping>
```

Figure 6.7: XML-Structure for mapping `PartOf`

Example 6.1 Going back to our ontology `PO` and its underlying database `PDB`, we attempt to represent the mapping between the relationship `PartOf` and its related extension in relation `Component`. The relation `Component` refers to the relation `Item1` using the foreign keys `SID`, `MID` to denote n-to-m relationships between items of products. We express this mapping in XML as shown in Figure 6.7. There, the target extension field, namely "PRODUCT.Name", is the same for $E_{projSub}$ and $E_{projObj}$. Notice that if n-to-m relationships are represented by more than two relations, the extension

fields will be different. The content of tags *<ProjectionFrom>* and *<ProjectionTo>* respectively refer to the subject and the object to which the `PartOf`-relationship is mapped. Each projection fragment includes the foreign key of relation `Component` and the attribute name of the referred relation `Item1`, namely 'Name'. Furthermore, we allow to introduce additional information about foreign keys in the mapping expression in order to enrich their descriptions. For instance, we can extend the XML-structure for the foreign key `MID` as depicted by Figure 6.8.

```
<DbAttribute name="MID">
        <ForeignKeyTo>
                <DbRelation name="PRODUCT">
                        <DbAttribute name="AID" />
                </DbRelation>
        </ForeignKeyTo>
</DbAttribute>
```

Figure 6.8: XML-Structure for Meta-Data of Foreign Keys

6.3 System Architecture and Technical Specifications

The architecture of the prototype ODBT is based on the concepts developed for semantic query transformation, as presented in this thesis. This section first describes the necessary components for implementing query transformation and then specifies them in more details. These components are:

1. A SQL-parser for parsing an input query expressed in SQL, extracting the query terms for transformation, and building a new query that takes ontological information into account.

2. An interface for communicating with different ontology tools. This interface will provide several functionalities such as storage, searching, and maintaining ontology data (cf. 6.1.3).

3. An interface for accessing mapping information. It will provide a set of functions to extract mapping elements from the mapping structures (cf. 6.2). It also has to specify correspondences for any cardinality of relationships.

4. A module that allows to build sub-expressions for the transformed query using the previous interface.

5. A user interface for editing user queries and displaying retrieved data.

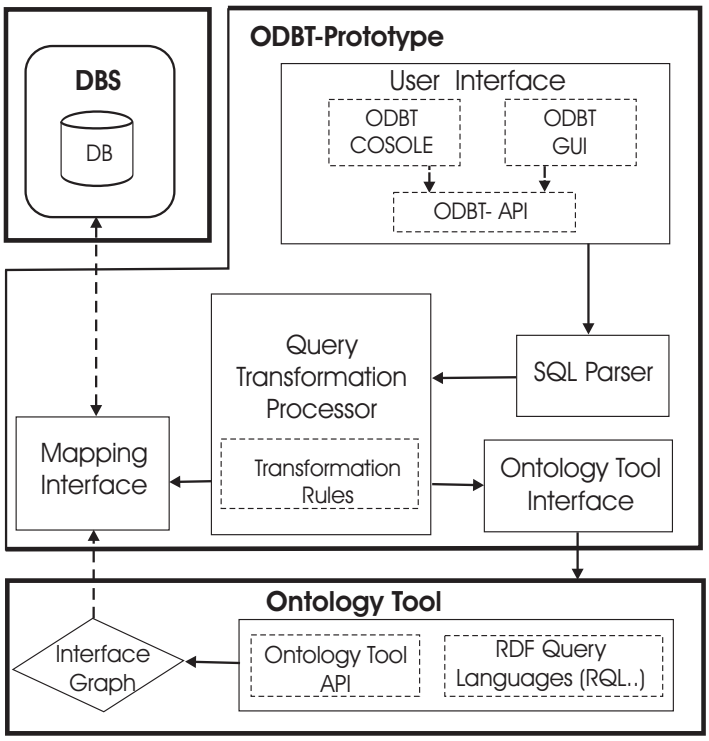


Figure 6.9: Access Diagram for Query Transformation

Based on these components an overall architecture of the system is presented in Figure 6.9. As we see from the figure, the system has three main parts: The *ODBT-prototype* which includes the components above, the *ontology tool* which manages ontology data, and the *database system* to which queries are posed. The arrows indicate access paths between the components. The access to the system is done using three interfaces: A *graphical-based* (GUI) or a *console-based interface*, through which users enter their SQL-queries and view the answers, and a *programming interface*, which provides the functionalities needed for this task. The *SQL-parser* parses queries expressed via the user interface, extracts and extends their expressions using terms provided by a *transformation processor*. The transformation processor constitutes the backbone of the query rewriting. It applies transformation rules to the parsed query by using mappings obtained from the mapping interface and ontology data obtained from the ontology tool. The ontology tool also enables the transformation processor to access inferred graphs of the underlying ontology. In its current version, the ODBT executes the rules independent of each others. The class diagram of the implementation is illustrated in Appendix C.

6.3.1 ODBT-API

The ODBT-prototype API for the ontology based query transformation is defined by the class `OdbtPrototyp`. An overview of this class is given in Appendix C (see Figure C.2). It provides the following functionalities:

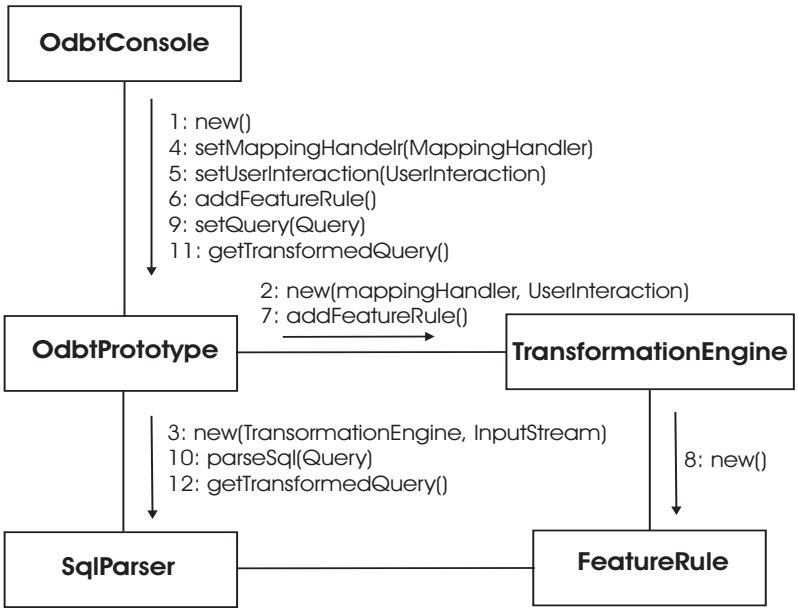


Figure 6.10: Interaction Diagram for ODBT-API Communication

- Interface for communicating with users
- Embedding of transformation rules
- Embedding of mappings
- Transformation of database queries

The interface for user communication allows a user, for example, to specify his context if homonymous terms are used in query expression (see Section 5.4.1). In this case, the interface offers the user several alternatives. The UML-interaction diagram in Figure 6.10 shows how a communication is established through ODBT-prototype API. It describes the different steps needed for transforming a query using FEATURE rule. These steps are enumerated. The corresponding class names and methods are denoted like the implemented ones. In step 6 the FEATURE rule has been set for query transformation. As result the class `TransformationEngine` creates an instance of class `FeatureRule`. The step 9 gives the rule to the class `OdbtPrototype`. This implies a call of function `parseSql` of the class `SqlParser`. This function extracts query terms and extends the query by means of the class `TransformationEngine`. Finally, the transformed query will be requested in step 11.

6.3.2 Mapping Interface

The class structure for the mappings has been designed in such way that it could be extended by several access and storage mechanisms. As shown in Figure 6.11 two interfaces are available for accessing the relevant storage structures. For our implementation the access to XML-mapping structures has been implemented, as described in Section 4.3. The class `CardinalityMappingHandler` uses not only methods of the

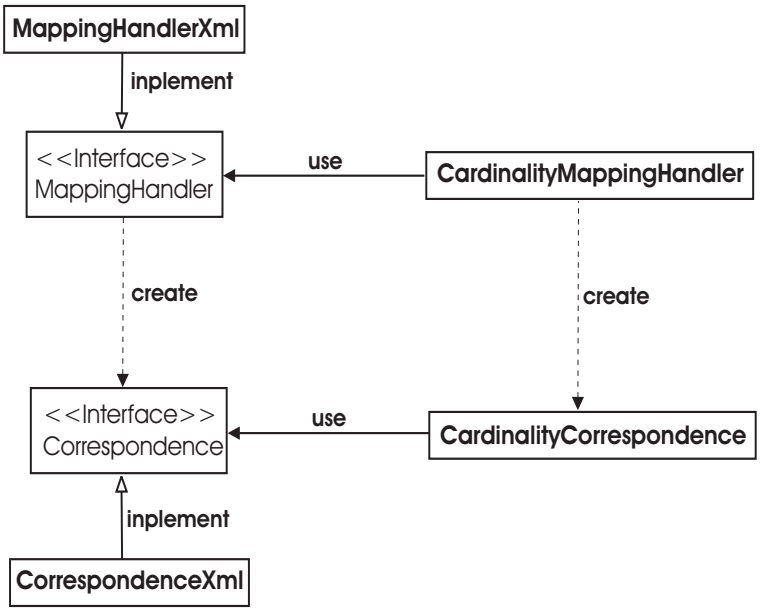


Figure 6.11: Class-Structure to access Mappings

`MappingHandler` interface but also additional methods that enable to find relevant correspondences for rule transformations. The class `CardinalityCorrespondence` contains methods for creating expressions for extending query conditions as well as expressions for restricting them. The methods designed for accessing mapping elements are provided by the `Correspondence` interface. An overview of all these methods is presented by an UML diagram in Appendix C (see Figure C.2).

The mapping interface provides information about the database and ontologies that are connected. These informations are represented in the mapping file where additional tags for the connection parameters are specified. In Appendix B.1 we show how these tags are defined.

6.3.3 Ontology Tool Interface

As previously mentioned we used Jena and Sesame for managing ontology data. Figure C.3 in Appendix C presents an UML class diagram that describes the access to ontology data as well as the building of inference rules by means of these tools. The class `OntologyToolHandler` builds the abstract basic class for communicating with ontology tools. Further, it is extended by the abstract class `OntologyToolAccess` using methods that enable access to ontology graphs, as described in in Section 6.1.3. The abstract class `OntologyToolAccess` also extends `OntologyToolHandler` class by introducing additional methods for inserting RDF/OWL data. We implemented the functionalities for these abstract classes in specific classes for both Jena and Sesame. For instance, we describe in the following how inference rules have been embedded in the prototype.

In Sesame, the configuration of SAIL (the Storage And Inference Layer) for the storage of RDF data is flexible. The configuration is carried out using the class

```
// Use of a database for the storage
RdfSchemaRepositoryConfig rdbmsSail = new
RdfSchemaRepositoryConfig(jdbcDriver,jdbcDNS,user,password);
// Specify a user-inference  rule
rdbmsSail.setInferencerClass
("org.openrdf.sesame.sailimpl.rdbms.CustomInferenceServices");
// Set the rule for SAIL
rdbmsSail.setParameter("rule-file",odbtRules);
```

Figure 6.12: Creation of SAIL for user-defined Rule in Sesame

```
//  A "ModellMaker" for relational database
ModelMaker maker = ModelFactory.createModelRDBMaker(dbConnection);
// Create an ontology model
Model odbtModel = maker.createModel("odbt");
// A "Reasoner" for user-defined inference rule
Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules (odbtRules));
// Create an inferred ontology model
InfModel odbtInfModel = ModelFactory.createInfModel(reasoner, odbtModel);
```

Figure 6.13: Creation of inferred Ontology Model in Jena

`RdfSchemaRepository- Config`. For our implementation, we use a MySql database to store ontologies and the connection is established through the JDBC-interface. To this end, information about the JDBC-driver, the DSN (Data Source Name), the user-name, and password should be transferred. User-defined rules can be configured by the method `setInferencerClass` and their related parameters can be set by the method `setParameter`, hence a new SAIL can be created in the 'Repository' of Sesame, as illustrated in Figure 6.12.

Similarly, we use a MySql database to store ontologies in Jena. The class `ModelMaker` enables to build several ontology models. In addition, the class `GenericRuleReasoner` allows users to specify their self-defined inference rules and the class `InfModel` provides the necessary methods to access their inferred data.

In Figure 6.13, we give an example of creating an inferred ontology model using a self-defined inference rule.

```
SELECT    x
FROM      {x}     <po:partOf> {pr:product}
USING NAMESPACE
                  po = <http://localhost/partof#>,
                  pr = <http://localhost/product#>
```

(a)

```
SELECT   ? x
WHERE ( ? x
            <http://localhost/partOf#partOf >
            <http://localhost/odbt#product#PC>)
```

(b)

Figure 6.14: Building Inference for concept `PC` in (a) Sesame and (b) Jena

In order to retrieve concepts and relationships from the stored ontologies, Sesame and Jena use SeRQL and RDQL languages, respectively. Figure 6.14 gives examples of querying Sesame and Jena ontologies to retrieve all concepts inferred by the `PartOf`-relationship for the concept `PC`. The use of namespace 'http://localhost/' denotes that

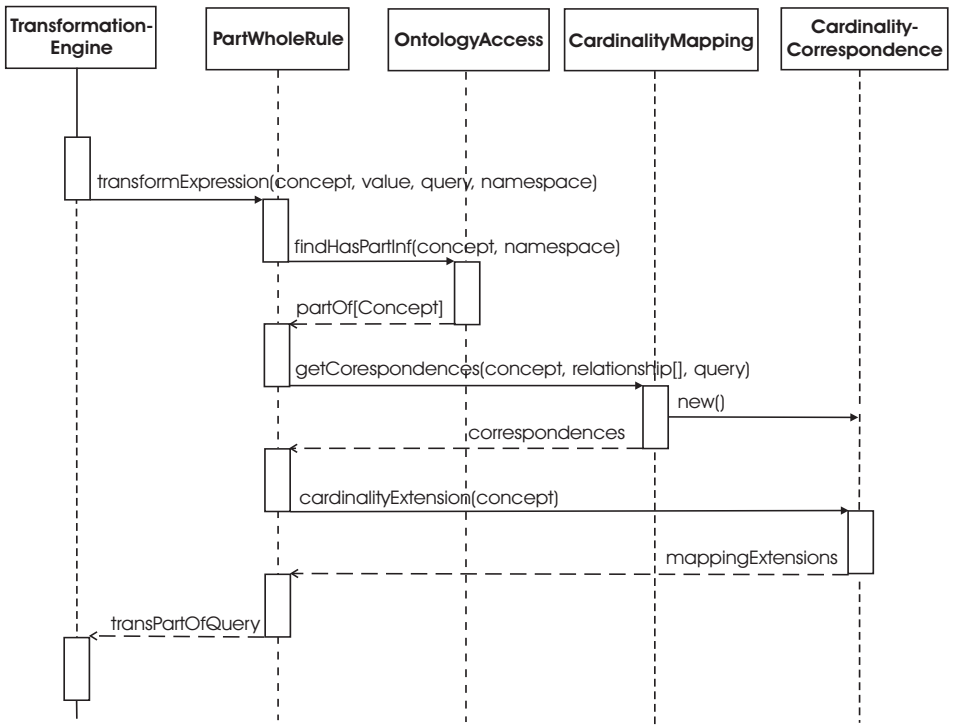


Figure 6.15: Interaction Diagram for PartWhole Rule

the concepts have been stored locally for test purposes. According to our experience, Sesame provides better performance than Jena in terms of response time when querying data inferred by `ISA-` or `PartOf` relationships.

6.3.4 Transformation Processor

The transformation processor performs the main tasks for query transformations. It utilizes information from database, ontology tool and mapping module in order to apply semantic rules. Appendix C (see Figure C.4) gives an overview of the class structure of the transformation processor.

We show now by means of a sequence diagram how the processor works for PartWhole rules (see Figure 6.15). Briefly, the method `transformExpression` allows to modify sub-expressions of the query by applying the rule with respect to values and their related concepts involved in the query. To this end, it calls the method `findHasPartOf` to find all relevant part concepts. The method `getCorrespondences` relies on defined mapping structures to get correspondences between the relationship `PartOf` and database attributes. Information about the type of correspondences are determined by the method `cardinalityExtension`. Accordingly, a new expression of query is formulated. Here, the word 'cardinality' refers to 1-to-1, 1-to-n, or n-to-m of relationships between objects as described in Sections 5.3 and 5.4. The manipulation of relevant correspondences is not directly performed by the rules, but they are set by the class `CardinalityMapping` as illustrated in the next section.

6.4 Prototype Tests

Our approach aims at improving the effectiveness rather than the efficiency of the retrieval. Efficiency is commonly measured in terms of the computer resources used such as memory and execution time. Effectiveness is a measure of the ability of the system to satisfy the user in terms of the relevance of tuples retrieved. Effectiveness is commonly measured in terms of *recall* and *precision* [BG94].

Formally, let A be the number of retrieved relevant tuples, B the number of relevant tuples not retrieved, and C be the number of irrelevant tuples retrieved. Following [BG94] we define recall and precision as follows.

Definition 6.1 (Recall) *For any given retrieved set of tuples, recall is the number of retrieved relevant tuples as a proportion of all relevant tuples in the database.*

$$Recall = \frac{A}{A+B}$$

Definition 6.2 (Precision) *For any given retrieved set of tuples, precision is the number of retrieved relevant tuples as a proportion of the number of retrieved tuples.*

$$Precision = \frac{A}{A+C}$$

Therefore, recall can be viewed as a measure of effectiveness in *selecting performance*, i.e., a measure of effectiveness in including relevant tuples in the retrieved set. Precision can be viewed as a measure of purity in retrieval performance, i.e., a measure of effectiveness in excluding irrelevant tuples from the retrieved set [BG94]. High precision and high recall are desirable. However, the computation of recall and precision require the determination of relevance in some way. Often, recall is estimated because it is often difficult to know how many relevant tuples exist in a database, especially if the database size is large. Obviously some tuples may be marginally relevant or somewhat irrelevant. Others may be very relevant and others may be completely irrelevant. In addition, different users may differ about the relevance or non-relevance of particular tuples to given request. In the field of information retrieval, the relevance assessments are usually made by a panel of experts in a particular discipline.

6.4.1 Effectiveness Evaluation

We implemented the ODBT-prototype in Java and currently consists of 15,426 lines of code. It is using `DB2` to store databases and `Mysql` to store ontologies for Jena and Sesame. All ontologies and their associated databases described in this thesis have been implemented and the example queries have been successfully transformed. As described in Section 6.3.2, we created XML-Mapping files for connecting the DBMS to the corresponding ontology tools. We must import such files in order to run the system. This can be done in two ways. By using the GUI interface, the file names are selected from the menu. By using the CONSOLE interface, the path name of the files must be entered as an additional parameter for the program calls.

The goal of implementing the ODBT-prototype is not only to test the applications described in this thesis but also to evaluate its effectiveness using an application in

the real world. To this end, we used the well known GNIS data set [5] which has been created by the "U.S. Geological Survey" and the "U.S. Board on Geographic Names". It provides data about social and cultural locations in each state in the USA. With these data, we populated a DB2 relation called Location. We also call the database GNIS. The relation has fourteen attributes and is populated by approximately two million entries. The main attributes of the relation are feature-name and feature-type. The attribute feature-name represents geographical or cultural features of locations, e.g., "river". The attribute feature-type represents the type of their features, e.g., "stream". We associated the WordNet ontology with the GNIS database. WordNet describes the meaning of many features. We did not adapted WordNet according to our assumptions as described in Section 4.3. However, we established one-to-one mappings between concepts of WordNet and database values. Furthermore, we mapped the relation name Location and the attribute name feature-name onto the WordNet concepts PHYSICAL THING and UNIT, respectively.

In the following, we examine how the ontology influences the effectiveness of the retrieval. To this end, we submitted three kinds of queries: QE_i, QS_i, and QT_i. We used values from the feature-type domain to retrieve all tuples matching values of the attribute feature-name. QEi queries select tuples whose values exactly match a given feature name f_i. They look like:

QE_i= SELECT * FROM Location WHERE feature-name= f_i

The QSi queries select tuples whose values inexactly match a given feature name f_i. We used the SQL-operator 'LIKE'. These queries look like:

QS_i= SELECT * FROM Location WHERE feature-name LIKE '%f_i%'

Finally, QT_i are queries obtained by applying the SYNONYMY and COLLECTION rules to queries QE_i.

Table 6.2 shows the result of the test queries for the following feature names: 'bay', 'bridge', 'channel', 'dam', 'island', 'sea', and 'stream'. We respectively denote by $QE_i(DB)$, $QS_i(DB)$, and $QT_i(DB)$ the number of tuples in the answers of QE_i, QS_i, and QT_i. We examined the answer of QS_i and QT_i, and calculated the number of tuples which we found correct and hence determined the precision measures [6] referred to as $P(QS_i)$ and $P(QT_i)$, respectively. We should mention that it is not possible to manually find all relevant tuples because of the big size of the database. Thus, we estimate the recall by identifying a pool of relevant tuples and then determining the proportion of the pool which the transformed query has retrieved. We create the pool by submitting queries for retrieving tuples where the values of attribute feature-type match the names above, then manually identifying the set of relevant tuples. We denote by $R(QS_i)$ and $R(QT_i)$ the recall of $QS_i(DB)$ and the recall of $QT_i(DB)$, respectively.

The test results show that the answers of QT_i-queries are better than the answers of QE_i-queries. Any initial query returns no results at all whereas its transformation provides results even with some loss of accuracy. Nevertheless, there are two reasons why

[5] http://geonames.usgs.gov/

[6] Expressed in percentage

f_i	$QE_i(DB)$	$QS_i(DB)$	$QT_i(DB)$	$P(QS_i)$	$P(QT_i)$	$R(QS_i)$	$R(QT_i)$
Bay	0	198	20	2.02 %	100 %	0.32%	1.62 %
Bridge	0	12	12	100 %	100 %	2.13 %	2.13 %
Channel	0	1	29	0 %	68.97 %	0 %	5.08 %
Dam	0	2321	3	4.30 %	100 %	1.84 %	0.06 %
Island	0	13	334	15.38 %	54.49 %	0.10 %	9.18 %
Sea	0	692	5	1.15 %	100 %	66.70 %	41.67 %
Stream	0	67	198	17.91 %	45.46 %	0.13 %	1.03 %

Table 6.2: Test Results

some of the transformed queries, e.g., QT_7, return irrelevant tuples. First, `WordNet` ontology uses some instance names to represent concepts. For example, it considers the island name 'Charles' as a sub-concept of the concept `ISLAND`. The `GNIS` database designers use such names to refer different things. For example, 'Charles' can be used to refer the river 'Charles' or a populated place in a city. Second, such names have not been deleted from the ontology as we do not need them for our query transformation. In our approach, we do not consider instances as part of an ontology. As a consequence, the mapping of concepts to database values was not as accurate as required. We conclude then that it is necessary to have correct mappings in order to retrieve the most relevant results.

In addition, the answers of QT_i-queries are better than the answers of QS_i-queries. QS_i-queries return many tuples, but most of them are irrelevant because the queries are based only on the syntax of feature names. Furthermore, all QT_i queries (except QT_4) return more relevant tuples than QS_i queries. Moreover, QT_i queries have low recall because the mapping is not complete, as illustrated in Section 4.4. That is, some `feature-name` values are not mapped to ontology concepts. The reason is that, the `WordNet` ontology does not capture the semantics of all geographical features of locations. For this application, the use of a domain ontology, which provide more specific semantics about the domain of geography, would improve the recall of QT_i. Nevertheless, using `WordNet` the queries QT_i have high precision. Furthermore, `WordNet` is well appropriate to cover semantics of the feature types since most of the values of `feature-type` are represented in that ontology.

6.4.2 Efficiency Evaluation

It is evident that the response time of a query increases if it is transformed. The main reason for this is that, the computation time is dominated by querying the ontology for finding concepts and their corresponding terms. Therefore, the response time heavily depends on the ontology tool being used to manage the ontologies. We performed a set of performance tests for the ontology tools, `Jena` and `Sesame`, used by ODBT.

For the tests we submitted a set of queries and transformed them using Collection Rule and measure the response time. The transformed queries have different number of equal predicates. Figure 6.16 shows that the response time increases as the number of the additional predicates in the transformed query increase. Additional predicates

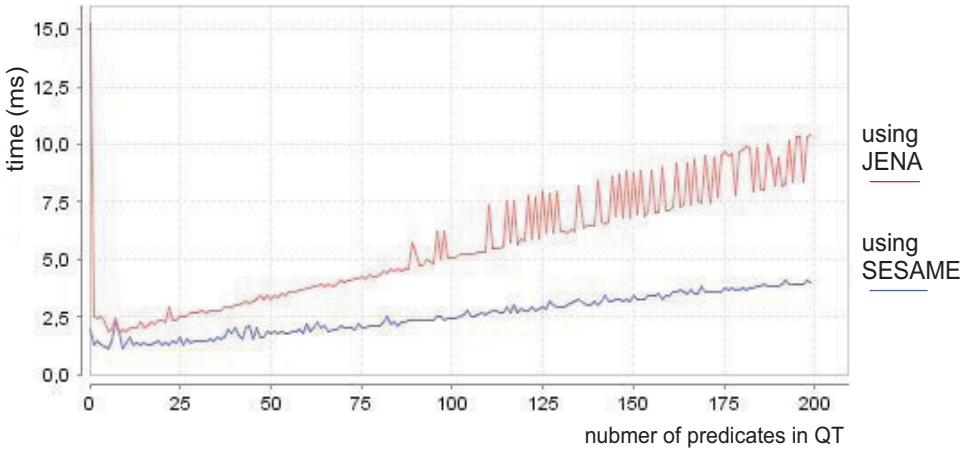


Figure 6.16: Run-time Performance of ODBT with Jena and Sesame

are built up using concepts retrieved from the ontology which are related through `ISA`-relationships. Compared to Jena, the (blue) curve of `Sesame` shows that ODBT with `Sesame` provides better run-time performance. In addition, by examining the (red) curve of `Jena` we deduce that the first query submitted to ODBT with `Jena` takes more time than the subsequent queries. The main reason is that, before executing the first query `Jena` has to build inferences in the main memory for the inferred graph, as illustrated in Section 6.1.3. However, `Sesame` does not need to build inferences during query execution since the inferred graph is stored in the database. Figure 6.17 depicts the result of measuring the amount of time spent on initializing Jena and Sesame. We consider different number of RDF-statements from the ontology. We deduce that for `Jena` the run-time increases with the number of statements whereas for `Sesame` it remains quite constant.

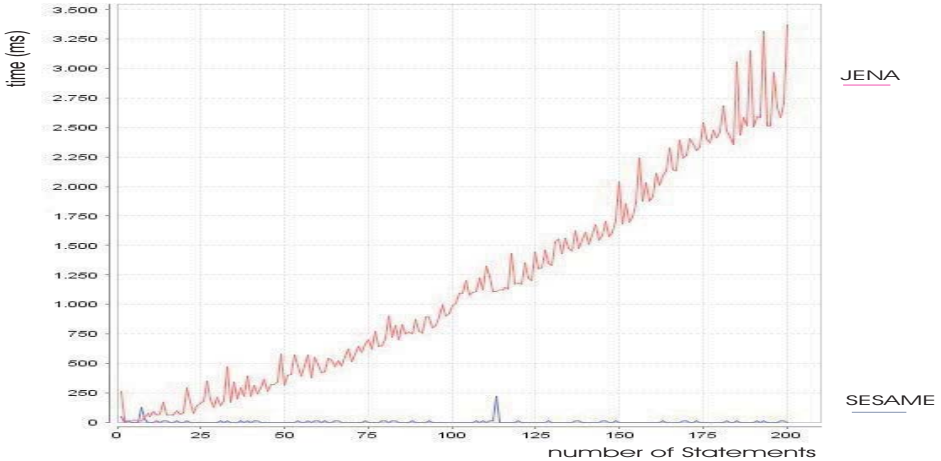


Figure 6.17: Run-time Initialization Phase

6.5 Summary

The ODBT-prototype shows that the concepts developed in this thesis are not just of theoretical interest but they provide practical solution to the semantic problem of query processing. In particular, we conclude that they can be put to work using existing Web technologies: Ontologies can be encoded using RDF/S schema, OWL can be used to support inference mechanisms, and Jena and Sesame tools are appropriate for storing and querying ontology data. Moreover, we used XML to develop a language for representing mapping structures required for query processing. The successful use of existing technologies makes us optimistic about the potential contribution of our approach to a more 'intelligent' query processing. We used the ODBT-prototype to evaluate the effectiveness of our semantic transformation for improving query processing. We successfully applied the prototype to query a geographic database in a semantical way. Our evaluation tests show that the semantic transformation provides more meaningful answers to user queries.

Chapter 7

Conclusion and Future Work

Extension of current database systems to support effective query processing has received a large amount of research attention, yet another challenge facing existing systems is the restrictive nature of query answering. That is, database systems support only exact query answering, i.e., they solely rely on exact syntax of queries to retrieve data. The meaning of real world objects represented in a database is neither explicitly specified nor completely captured by these systems. As consequence, query answering often does not meet user intentions. This problem has not received much attention in database research. The main contribution of this work is to extend existing database systems to deal with the meaning of queries. We present a new approach to improve query answering based on semantics from ontologies. User queries are transformed into queries which can provide more meaningful results, better meeting the intention of the user. We conclude that ontologies are indeed a suitable technology for supporting the semantic transformation. More specifically, the major contributions of this thesis are as follows:

1. The effective use of ontologies for enhancing query processing in database management systems. We show how semantics of objects represented in a database can be explicitly captured and used to provide meaningful results for user queries.

2. The definition and specification of different kinds of mappings that relate concepts of an ontology with those of a relational database. We consider both extension and schema of the database for the mapping specification. However, current work in this area ignores the mapping of database extensions. We developed a set of algorithms to determine such mappings based on linguistic, structural and semantic approaches.

3. The development of two sets of transformation rules for the semantic transformation. The first set extends the answer of a query with meaningful results that are relevant to the user. The second set restricts the answer of a query by reducing non-meaningful results that are irrelevant. We demonstrate the usefulness of the rules based on practical applications.

4. The successful use of the theory of term rewriting systems to formalize the query transformation process and to study basic properties for the rules application.

The transformation process is correct and we prove that it terminates. In addition, we distinguish two sets of rules according to their application. We prove that the answer of the transformed query produced by rules of the first set does not depend on the application order. We also prove that the order of rules of the second set may affect the quality of the answer, and hence some irrelevant tuples might be returned. However, we demonstrate how we can obtain a meaningful transformation even by applying all the rules.

5. The building of the prototype system `ODBT` which implements the ontology-based approach we advocate for the semantic query transformation. We show how we can use available technologies for implementing some components of the prototype. We also develop an XML syntax-based language for representing mapping information. The language is based on our mapping specification with the focus on the mapping of database extensions.

6. The evaluation of the capability of our semantic approach by implementing an application over a real-word database and a real-world ontology. The results of some test queries submitted to the database indicate that our semantic transformation is able to provide answers with high levels of user satisfaction.

Our approach is useful for building scientific applications in several domains such as geographical information systems, medical systems, and biological systems. For this purpose, some prominent ontologies can be employed, such as the Gene Ontology [Con06]. Furthermore, with the realization of the Semantic Web more and more domain ontologies will become available in the near feature.

There are many possible directions for future research on semantic query processing due to the importance of the problem. In the following, we propose improvements to our approach from three different viewpoints: ontology development, mapping discovery, and query answering.

1. Building and reusing ontologies: A well-defined ontology that describes the underlying database is fundamental for applying the semantic transformation. Whenever an ontology is not available, methodologies and methods are required to enable the generation of ontologies. Building an ontology that completely captures the semantics of database is of great interest. To this end, an important area of research is finding potential sources to extract semantics that are embedded in the database, e.g., meta-data, schema [JDM02]. Reverse-engineering techniques can help to semi-automatically generate ontologies [Ast05]. When an ontology is already available, we need methodologies and methods to efficiently manage, adapt, and control its great amount of concepts for the purpose of reuse (building a new ontology rather than starting from scratch) [UHW+98]. There, the major task is to identify or create concepts and relationships that describe database schema and values. Data mining and clustering techniques can support this task. For example, techniques based on thresholding can help to identify ontology concepts related to the underlying data [EM01]. Moreover, we need tools that facilitate extraction of semantics, evaluation tools, and tools for editing and

processing ontology data. Positive results in this area will motivate the use of ontologies.

2. Developing advanced approaches for the mapping problem: Finding the mapping between ontologies and relational databases deserves special research attention. Currently, we are not aware of work that has addressed this problem. Existing mapping systems, which either perform schema-to-schema mapping or ontology-to-ontology mapping, should be adapted for performing schema-to-ontology mapping and extension-to-ontology mapping. The latter mappings have distinctive features which must be considered such as taxonomies, complex semantic relationships, and absence of ontology instances. Furthermore, we need mapping systems that reduce user interventions during the mapping discovery. The deployment of an ontology for query transformation would be greatly supported by more automated systems.

3. Generating intentional answers: The transformation approach can be extended to provide users with intentional answers. An intentional answer can be seen as a set of statements that characterize the query results for some relevant attributes [HHCF96]. An example is "20 % of electronic products are computers". Intentional answers are meaningful to the user, especially when the answer to his query is empty or includes a large number of tuples. Traditional approaches to intentional query answering intend to build generalizations of the attribute values. Based on these generalizations and the number of tuples in the answer, statements can be generated. However, building such generalizations may be error-prone and may not precisely describe the meaning of values. We suggest the use of ISA-taxonomy of the ontology to generalize concepts that correspond to the attribute values. Generalizations can then be done by ascending the taxonomy and retrieve higher concepts. Data mining techniques including the attribute-oriented induction can be used to support the generation of intentional answers in relational database systems, as described in the work of Yoon [YSP97].

Bibliography

[AGG02] Ashwin, T.V.; Gupta, R.; Ghosal, S.: Adaptable similarity search using non-relevant information. In: *Proceedings of the 28th international conference on Very Large Data Bases (VLDB'02)*, pp. 47–58. 2002.

[AHK01] Arens, Y.; Hsu, C.; Knoblock, C.A.: Query processing in the SIMS information mediator. In Readings in agents. Morgan Kaufmann Publishers Inc., 2001.

[And92] Andrews, T.: The ONTOS object database. In: *DM91: Conference proceedings on Data Management 91*, pp. 33–36. 1992.

[ART96] Asirelli, P.; Renso, C.; Turini, F.: Language extensions for semantic integration of deductive databases. In: *Logic in Databases*, pp. 415–434. 1996.

[Ast05] Astrova, I.: Towards the Semantic Web- An approach to reverse engineering of relational databases to ontologies. In: *ADBIS Research Communications*, pp. 121–147. 2005.

[BB03] Borgida, A.; Brachman, R.J.: Conceptual modeling with description logics. The Description Logic Handbook - Theory, Implementation and Applications. pages: 349-372. Cambridge University Press, 2003.

[BBB+97] Bayardo, R.J.; Bohrer, W.; Brice, R.; Cichocki, A.; Fowler, J.; Helal, A.; Kashyap, V.; Ksiezyk, T.; Martin, G.; Nodine, M.; Rashid, M.; Rusinkiewicz, M.; Shea, R.; Unnikrishnan, C.; Unruh, A.; Woelk, D.: InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 195–206. 1997.

[BBC+00] Beneventano, D.; Bergamaschi, S.; Castano, S.; Corni, A.; Guidetti, R.; Malvezzi, G.; Melchiori, M.; Vincini, M.: Information integration: The MOMIS project demonstration. In: *VLDB' 00*, pp. 611–614. 2000.

[BBM+92] Brachman, R.J.; Borgida, A.; McGuinness, D.L.; Patel-Schneider, P.F.; Resnick, L.A.: The CLASSIC knowledge representation system or, KL-ONE: The next generation. In: *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS)*, pp. 1036–1043. 1992.

[BBS03] Beneventano, D.; Bergamaschi, S.; Sartori, C.: Description logics for semantic query optimization in object-oriented database systems. ACM Transaction on Database Syststems. pages: 1-50, 2003.

[BCM02] Binderberger, M.O.; Chakrabarti, K.; Mehrotra, S.: An approach to integrating query refinement in SQL. In: *Proceedings of the 8th International Conference on Extending Database Technology (EDBT'02)*, pp. 15–33. 2002.

[BCN+03] Baader, F.; Calvanese, D.; Nardi, D.; McGuinness, D.; Patel-Schneider, P.: *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, New Jersey, USA, 2003.

[BCP04] Barrasa, J.; Corcho, O.; Pérez, A.G.: R2O, an extensible and semantically based database-to-ontology mapping language. In Second Workshop on Semantic Web and Databases (SWDB2004). pages: 123-138. Toronto, Canada., 2004.

[BDH+95] Buneman, P.; Davidson, S.B.; Hart, K.; Overton, G.C.; Wong, L.: A data transformation system for biological data sources. In: *VLDB*, pp. 158–169. 1995.

[BDK92] Bancilhon, F.; Delobel, C.; Kanellakis, P., editors: *Building an object-oriented database system: the story of 02*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 1992.

[Bee04] Beeferman, D.: Onelook: English dictionary. Web page available at: http://www.onelook.com, 2004.

[BG94] Buckland, M.; Gey, F.: The relationship between recall and precision. In: *Journal of the American Society for Information Science*, volume 45(1):pp. 12–19, 1994.

[BG06] Brickley, D.; Guha, R.V.: RDF vocabulary description language 1.0: RDF schema. (W3C recommendation 10 february 2004). Web page available at: http://www.w3.org/TR/rdf-schema/, 2006.

[Bin02] Binderberger, M.O.: *Integrating Similarity Based Retrieval and Query Refinement in Databases*. Ph.D. thesis, UIUC - University of Illinois at Urbana-Champaign, Urbana, Illinois, 2002.

[Biz03] Bizer, C.: D2R MAP- A database to RDF mapping language. In 12th Int. World Wide Web Conference. pages: 102-121. Budapest. May, 2003.

[BKvH02] Broekstra, J.; Kampman, A.; van Harmelen, F.: Sesame: A generic architecture for storing and querying RDF and RDF schema. In: *International Semantic Web Conference*, pp. 54–68. 2002.

[BLHL01] Berners-Lee, T.; Hendler, J.; Lassila, O.: The semantic web, a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. In: *Scientific American*. 2001.

[BM04] Borgida, A.; Mylopoulos, J.: Data semantics revisited. In: *SWDB*, pp. 9–26. 2004.

[BN98] Baader, F.; Nipkow, T.: *Term Rewriting and All That*. Cambridge University Press, United Kingdom, 1998.

[Bor97] Borst, W.N.: *Construction of Engineering Ontologies For knowledge sharing and reuse*. Ph.D. thesis, University of Twente, Enschede, 1997.

[BP02] Beck, H.; Pinto, H.S.: Overview of approach, methodologies, standards, and tools for ontologies. In: , 2002.

[Bri94] Brill, E.: Some advances in transformation-based part of speech tagging. In: *AAAI*, pp. 722–727. 1994.

[Bry89] Bry, F.: Towards an efficient evaluation of general queries: Quantifier and disjunction processing revisited. In: *SIGMOD Records*, volume 18(2):pp. 193–204, 1989.

[BSZ03] Bouquet, P.; Serafini, L.; Zanobini, S.: Semantic coordination: A new approach and an application. In: *International Semantic Web Conference*, pp. 130–145. 2003.

[BSZ04] Bouquet, P.; Serafini, L.; Zanobini, S.: Peer-to-peer semantic coordination. In: *In Journal of Web Semantics*, volume 2(1):pp. 81–97, 2004.

[Bub80] Bubenko, J.A.: Information modeling in the context of system development. In: *IFIP Congress*, pp. 395–411. 1980.

[BW77] Bobrow, D.G.; Winograd, T.: On overview of KRL, a knowledge representation language. In: *Cognitive Science*, volume 1(1):pp. 3–46, 1977.

[BY85] Bravoco, R.R.; Yadav, S.B.: A methodology to model the dynamic structure of an organization. In: *Information Syststems*, volume 10 (3):pp. 299–317, 1985.

[CAdV01] Castano, S.; Antonellis, V. De; di Vimercati, S.: Global viewing of heterogeneous data sources. In: *IEEE Transactions on Knowledge and Data Engineering*, volume 13(2):pp. 277–297, 2001.

[CBS94] Chiang, R.H.L.; Barron, T.M.; Storey, V.C.: Reverse engineering of relational databases: Extraction of an EER model from a relational database. In: *Data Knowl. Eng.*, volume 12(2):pp. 107–142, 1994.

[CFF+98] Chaudhri, V.K.; Farquhar, A; Fikes, R.; Karp, P.D.; Rice, J.: Okbc: A programmatic foundation for knowledge base interoperability. In: *AAAI/IAAI*, pp. 600–607. 1998.

[CFP82] Casanova, M.A.; Fagin, R.; Papadimitriou, C.H.: Inclusion dependencies and their interaction with functional dependencies. In: *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems (PODS '82)*, pp. 171–176. 1982.

[CGK+99] Cheng, Q.; Gryz, J.; Koo, F.; Leung, T.Y.C.; Liu, L.; Qian, X.; Schiefer, K.B.: Implementation of two semantic query optimization techniques in DB2 universal database. In: *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pp. 687–698. San Francisco, USA, 1999.

[CGM90] Chakravarthy, U.; Grant, J.; Minker, J.: Logic-based approach to semantic query optimization. In: *ACM Transansaction on Database Systems*, volume 15(2):pp. 162–207, 1990.

[Cha98] Chaudhuri, S.: An overview of query optimization in relational systems. In: *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS'98)*, pp. 34–43. New York, NY, USA, 1998.

[Che80] Chen, P. P., editor: *Entity-Relationship Approach to Systems Analysis and Design. Proc. 1st International Conference on the Entity-Relationship Approach*. North-Holland, 1980.

[CJB99] Chandrasekaran, B.; Josephson, J.R.; Benjamins, V.R.: What are ontologies, and why do we need them? In: *IEEE Intelligent Systems*, pp. 20–26. 1999.

[CLN99] Calvanese, D.; Lenzerini, M.; Nardi, D.: Unifying class-based representation formalisms. In: *Journal on Artificial Intelligence Res. (JAIR)*, volume 11:pp. 199–240, 1999.

[Cod72] Codd, E.F.: Relational completeness of data base sublanguages. In: *In R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.

[Com03a] Company, ECLASS: ECLASS: The electronic classification. Web page available at: http://www.eclass.org, 2003.

[Com03b] Company, UNSPSC: UNSPSC: The united nations standard products and services code. Web page available at: http://www.unspsc.org, 2003.

[Con06] Consortium, Gene Ontology: The Gene Ontology. Web page available at: http://www.geneontology.org, 2006.

[Cov99] Cover, R.: Meta data coalition. open information model, version 1.0. Web Page Available at: http://mdcinfo/oim/oim10.html, 1999.

[CPCF04] Corcho, Ó.; Pérez, A.G.; Cabero, R.G.; Figueroa, M.C.S.: ODEVAL: A tool for evaluating RDF(S), DAML+OIL and OWL concept taxonomies. In: *AIAI*, pp. 369–382. 2004.

[CV92] Chaudhuri, S.; Vardi, M.Y.: On the equivalence of recursive and nonrecursive datalog programs. In: *Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 55–66. 1992.

[CW93] Ceri, S.; Widom, J.: Managing semantic heterogeneity with production rules and persistent queues. In: *VLDB*, pp. 108–119. 1993.

[CX95] Croft, W.B.; Xu, J.: Corpus-specific stemming using word form co-occurence. In: *Proceedings for the Fourth Annual Symposium on Document Analysis and Information Retrieval*, pp. 147–159. Las Vegas, 1995.

[CZ96] Cherniack, M.; Zdonik, S.B.: Rule languages and internal algebras for rule-based optimizers. In: *SIGMOD Conference*, pp. 401–412. 1996.

[CZ98] Cherniack, M.; Zdonik, S.: Changing the rules: Transformations for rule-based optimizers. In: *SIGMOD Rec.*, volume 27(2):pp. 61–72, 1998.

[Dat02] Date, C.J.: *Introduction to Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 2002.

[DDH01] Doan, A.; Domingos, P.; Halevy, A.Y.: Reconciling schemas of disparate data sources: A machine-learning approach. In: *SIGMOD' 01*, pp. 509–520. 2001.

[DGH+96] Daruwala, A.; Goh, C. Hian; Hofmeister, S.; Hussein, K.; Madnick, S.E.; Siegel, M.: The context interchange network prototype. In: *Proceedings of the Sixth IFIP TC-2 Working Conference on Data Semantics (DS-6)*, pp. 65–92. 1996.

[DHL+04] Dorneles, C.F.; Heuser, C.A.; Lima, A.E.N.; Silva, A.S.; Moura, E.S.: Measuring similarity between collection of values. In: *Proceedings of the 6th annual ACM international workshop on Web information and data management*, pp. 56–63. 2004.

[Dic00] Dictionaries, American Heritage: *The American Heritage Dictionary of the English Language*. Houghton Mifflin; 4 edition, BOSTON: HOUGHTON MIFFLIN, 2000.

[DS89] Derrett, N.; Shan, M.C.: Rule-based query optimization in iris. In: *CSC '89: Proceedings of the 17th conference on ACM Annual Computer Science Conference*, pp. 78–86. 1989.

[EL04] Efraty, N.; Landau, G.M.: Sparse normalized local alignment. In: *CPM*, pp. 333–346. 2004.

[EM01] E. Mena, A. Illarramendi: *Ontology-Based Query Processing for Global Information Systems*. Kluwer Publisher, USA, 2001.

[ES04] Ehrig, M.; Sure, Y.: Ontology mapping- An integrated approach. In: *ESWS*, pp. 76–91. 2004.

[Fel98] Fellbaum, C.: *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[FFR96] Farquhar, A.; Fikes, R.; Rice, J.: The ontolingua server: A tool for collaborative ontology construction. Technical report, Stanford KSL, pages: 96-26, 1996.

[FHvH+00] Fensel, D.; Horrocks, I.; van Harmelen, F.; Decker, S.; Erdmann, M.; Klein, M.C.A.: OIL in a nutshell. In: *EKAW*, pp. 1–16. 2000.

[FL01] Fung, C.; Li, Q.: Efficient multimedia database indexing using structural Join index hierarchy. In: *IEEE Pacific Rim Conference on Multimedia*, pp. 359–366. 2001.

[FMV93] Freytag, J.C.; Maier, D.; Vossen, G.: *Query Processing for Advanced Database Systems*. Morgan Kaufmann Publishers Inc., 1993.

[Fre85] Freytag, J.C.: *Translating relational queries into iterative programs*. Ph.D. thesis, Harvard University, 1985.

[Fre87] Freytag, J.C.: A rule-based view of query optimization. In: *SIGMOD Conference*, pp. 173–180. 1987.

[GBMS99] Goh, C.H.; Bressan, S.; Madnick, S.E.; Siegel, M.: Context interchange: New features and formalisms for the intelligent integration of information. In: *ACM Trans. Inf. Syst.*, volume 17(3), 1999.

[GD87] Graefe, G.; DeWitt, D.J.: The EXODUS optimizer generator. In: *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pp. 160–172. 1987.

[Gen91] Genesereth, M.R.: Knowledge interchange format. In: *KR*, pp. 599–600. 1991.

[GG95] Guarino, N.; Giaretta, P.: Ontologies and knowledge bases: towards a terminological clarification. In: *Knowledge Building Knowledge Sharing,ION Press*, pp. 25–32. 1995.

[GG+97] Grant, J.; Gryz, J.; ; Minker, J.; Raschid, L.: Semantic query optimization for object databases. ICDE, November 1997.

[GGCD02] Giboin, A.; Gandon, F.; Corby, O.; Dieng, R.: Assessment of ontology-based tools: A step towards systemizing the scenario approach. In: *13th InternationalConference on Knowledge Engineering and Knowledge Management*, pp. 63–73. 2002.

[GMB94] Greenspan, S.J.; Mylopoulos, J.; Borgida, A.: On formal requirements modeling languages: RML revisited. In: *ICSE*, pp. 135–147. 1994.

[GR04] Gutiérrez, M.; Rodríguez, A.: Querying heterogeneous spatial databases: Combining an ontology with similarity functions. In: *S. Wang et al (eds.), ER Workshop on Conceptual Modeling of GIS 2004*, pp. 160–171. 2004.

[Gra94] Graefe, G.: Volcano: An extensible and parallel query evaluation system. In: *IEEE Transactions on Knowledge and Data Engineering*, volume 6(1):pp. 120–135, 1994.

[Gro03] Grosse, J.: Speicherverfahren und werkzeuge fuer RDF/S. XML Clearinghouse Report, 2003.

[Gro05] Group, OMG Object Management: UML: The unified modeling language. UML Resource page. Web page available at: http://www.uml.org, 2005.

[Gru93] Gruber, T.R: A translation approach to portable ontology specifications. In: *Knowledge Acquisition (5) No. 2, USA*, pp. 199–220. 1993.

[GSN+01] Goble, C.A.; Stevens, R.; Ng, G.; Bechhofer, S.; Paton, N.W.; Baker, P.G.; Peim, M.; Brass, A.: Transparent Access to Multiple Bioinformatics Information Sources. In: *IBM Systems Journal Special issue on deep computing for the life sciences*, volume 40(2):pp. 532 – 552, 2001.

[HAI02] Hunt, E.; Atkinson, M.P.; Irving, R.W.: Database indexing for large DNA and protein sequence collections. In: *VLDB J.*, volume 11(3):pp. 256–271, 2002.

[HBEV04] Haase, P.; Broekstra, J.; Eberhart, A.; Volz, R.: A comparison of RDF query languages. In: *International Semantic Web Conference*, pp. 502–517. 2004.

[HFLP89] Haas, L.M.; Freytag, J.C.; Lohman, G.M.; Pirahesh, H.: Extensible query processing in Starburst. In: *SIGMOD Conference*, pp. 377–388. 1989.

[HHCF96] Han, J.W.; Huang, Y.; Cercone, N.; Fu, Y.J.: Intelligent query answering by knowledge discovery techniques. In: *IEEE Transactions on Knowledge And Data Engineering*, volume 8, pp. 373–390. 1996.

[HHL03] Heflin, J.; Hendler, J.A.; Luke, S.: SHOE: A blueprint for the semantic Web. In: *Spinning the Semantic Web*, pp. 29–63. 2003.

[HK96] Hsu, C-N.; Knoblock, C.A.: Using inductive learning to generate rules for semantic query optimization. In: *Advances in Knowledge Discovery and Data Mining*, pp. 425–445. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[HM96] Hammer, M.; McLeod, D.: Database description with SDM: A semantic database model. Readings in database systems, 1996.

[HMH01] Hernández, M.A.; Miller, R.J.; Haas, L.M.: Clio: A semi-automatic tool for schema mapping. In: *Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD '01)*, p. 607. 2001.

[Hor02] Horrocks, I.: DAML+OIL: A reason-able Web Ontology language. In: *EDBT*, pp. 2–13. 2002.

[HR95] Hacid, M.; Rigotti, C.: Combining resolution and classification for semantic query optimization in dood. In: *DOOD*, pp. 447–466. 1995.

[Hsu96] Hsu, C.: *Learning effective and robust knowledge for semantic query optimization*. Ph.D. thesis, UNIVERSITY OF SOUTHERN CALIFORNIA, 1996.

[Hue80] Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. In: *J. ACM*, volume 27(4):pp. 797–821, 1980. ISSN 0004-5411. doi:http://doi.acm.org/10.1145/322217.322230.

[HZ80] Hammer, M.; Zdonik, S.B.: Knowledge-based query processing. In: *VLDB*, pp. 137–147. 1980.

[Ioa96] Ioannidis, Y.E.: Query optimization. In: *ACM Comput. Surv.*, volume 28(1):pp. 121–123, 1996. ISSN 0360-0300.

[Jan01] Jannink, J.: *A Word Nexus for Systematic Interoperation of Semantically Heterogeneous Data Sources*. Ph.D. thesis, Stanford University, 2001.

[JCV84] Jarke, M.; Clifford, J.; Vassiliou, Y.: An optimizing prolog front-end to a relational query system. In: *SIGMOD Conference*, pp. 296–306. 1984.

[JDM02] Jarrar, M.; Demey, J.; Meersman, R.: On using conceptual data modeling for ontology engineering. In: *Journal on Data Semantics, Special issue on Best papers from the ER/ODBASE/COOPIS 2002 Conferences. LNCS,*, volume 2800(1):pp. 185–207, 2002.

[JK84] Jarke, M.; Koch, J.: Query optimization in database systems. In: *ACM Comput. Surv.*, volume 16(2):pp. 111–152, 1984.

[JMP97] Jhingran, A.; Malkemus, T.; Padmanabhan, S.: Query optimization in DB2 parallel edition. In: *IEEE Data Eng. Bull.*, volume 20(2):pp. 27–34, 1997.

[KC01] Kayed, A.A.; Colomb, R.M.: Extracting ontological concepts for tendering conceptual structures. In: *Data and Knowledge Engineering*, volume 41(4), 2001.

[KC04] Klyne, G.; Carroll, J.: Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation. Web page available at: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, 2004.

[KCT06] Karp, P.D.; Chaudhri, V.K.; Thomerne, J.: XOL: An XML-based Ontology exchange language. (version 0.4 august 31, 1999). Web page available at: http://www.ai.sri.com/ pkarp/xol/xol.html/, 2006.

[KE01] Kemper, A.; Eickler, A.: *Datenbanksysteme - Eine Einführung, 4. Auflage*. Oldenbourg, 2001.

[Kin81] King, J.J.: QUIST: A system for semantic query optimization in relational databases. In: *VLDB*, pp. 510–517. 1981.

[KLK91] Krishnamurthy, R.; Litwin, W.; Kent, W.: Language features for interoperability of databases with schematic discrepancies. In: *SIGMOD '91: Proceedings of the 1991 ACM SIGMOD international conference on Management of data*, pp. 40–49. 1991.

[KLW95] Kifer, M.; Lausen, G.; Wu, J.: Logical foundations of object-oriented and frame-based languages. In: *J. ACM*, volume 42(4):pp. 741–843, 1995.

[KM90] Kemper, A.; Moerkotte, G.: Advanced query processing in object bases using access support relations. In: *VLDB '90: Proceedings of the 16th International Conference on Very Large Data Bases*, pp. 290–301. San Francisco, USA, 1990.

[Kos00] Kossmann, D.: The state of the art in distributed query processing. In: *ACM Computing Survey*, volume 32(4):pp. 422–469, 2000.

[KS91] Kim, W.; Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. In: *IEEE Computer*, volume 24(12):pp. 12–18, 1991.

[Lau03] Lauser, B.: Semi-automatic ontology engineering and ontology supported document indexing in a multilingual environment. Diploma thesis, Universität Fridericiana Karlsruhe (TH), Germany, 2003.

[LB02] LOPES, P.S.N.D.; BARRETO, M.R.P.: Requirements modeling with UML discussed. Technical report, SENAC Faculty of Sciences and Technologies, 2002.

[LG90] Lenat, D.B.; Guha, R.V.: *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, Reading, Massachusetts, 1990.

[LH89] Lee, S.; Han, J.: Semantic query optimization in recursive databases. In: *Proceedings of the IEEE International Conference on Data Engineering*, pp. 444–454. IEEE Computer Society, 1989.

[LH91a] Lakshmanan, L.V.S.; Hernandez, H.J.: Structural query optimization: A uniform framework for semantic query optimization in deductive databases. In: *Proceedings of the 10th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 102–114. 1991.

[LH91b] Lee, S.H.; Henschen, L.J.: Evaluation of extended recursive queries in deductive databases. In: *DASFAA*, pp. 209–215. 1991.

[LHGP99] Liu, L.; Halper, M.; Geller, J.; Perl, Y.: Controlled vocabularies in OODBs: Modeling issues and implementation. In: *Distributed and Parallel Databases*, pp. 37–65. 1999.

[LHQ91] Lee, S.; Henschen, L.J.; Qadah, G.Z.: Semantic query optimization in deductive databases. In: *Proceedings of the IEEE International Conference on Data Engineering*, pp. 232–239. Kobe, Japan, 1991.

[Lin98] Lin, D.: An information-theoretic definition of similarity. In: *ICML '98: Proceedings of the 15th International Conference on Machine Learning*, pp. 296–304. San Francisco, USA, 1998.

[LM77] Levesque, H.J.; Mylopoulos, J.: An overview of a procedural approach to semantic networks. In: *IJCAI*, p. 283. 1977.

[LM92] Lakshmanan, L.V.S.; Missaoui, R.: On semantic query optimization in deductive databases. In: *IEEE International Conference on Data Engineering*, pp. 368–375. 1992.

[LM95] Lakshmanan, L.V.S.; Missaoui, R.: Pushing semantics inside recursion: A general framework for semantic optimization of recursive queries. In: *ICDE*, pp. 211–220. 1995.

[LNR97] Lee, A.J.; Nica, A.; Rundensteiner, E.A.: Keeping virtual information resources up and running. In: *CASCON '97: Proceedings of the 1997 conference of the Center for Advanced Studies on Collaborative research*, p. 13. 1997.

[LP82] Louis, G.; Pirotte, A.: A denotational definition of the semantics of DRC, a domain relational calculus. In: *VLDB*, pp. 348–356. 1982.

[LS95] Levy, A.Y.; Sagiv, Y.: Semantic query optimization in datalog programs. In: *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 163–173. 1995.

[LS03] Lerner, A.; Shasha, D.: A query language for ordered data, optimization techniques, and Experiments. In: *VLDB*, pp. 345–356. 2003.

[LWZ04] Law, Y.; Wang, H.; Zaniolo, C.: Query languages and data models for database sequences and data streams. In: *VLDB*, pp. 492–503. 2004.

[Mac91] MacGregor, Robert M.: Inside the LOOM description classifier. In: *SIGART Bulletin*, volume 2 (3):pp. 88–92, 1991.

[MBR01] Madhavan, J.; Bernstein, P.A.; Rahm, E.: Generic schema matching with Cupid. In: *VLDB*, pp. 49–58. 2001.

[McB02] McBride, B.: Jena: A Semantic Web toolkit. In: *IEEE Internet Computing*, volume 6(6):pp. 55–59, 2002.

[MD03] Malladi, R.; Davis, K. C.: Applying multiple query optimization in mobile databases. In: *HICSS*, p. 294. 2003.

[ME96] Monge, Alvaro E.; Elkan, Charles: The field matching problem: Algorithms and applications. In: *Knowledge Discovery and Data Mining*, pp. 267–270. 1996.

[Mee01] Meersman, R.: Ontologies and databases: More than a fleeting resemblance. In OES/SEO Workshop Rome, 2001.

[Mel95] Melamed, I.D.: Automatic evaluation and uniform filter cascades for inducing N-best translation lexicons. In: Yarovsky, David; Church, Kenneth, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pp. 184–198. Somerset, USA, 1995.

[MGMR02] Melnik, S.; Garcia-Molina, H.; Rahm, E.: Similarity Flooding: A versatile graph matching algorithm and its application to schema matching. In: *ICDE' 02*, pp. 117–128. 2002.

[MH06] McGuinness, D.L.; Harmelen, F.: OWL: Web ontology language overview. (W3C recommendation 10 february 2004). Web page available at: http://www.w3.org/TR/owl-features/, 2006.

[Min74] Minsky, M.: A framework for representing knowledge. Technical report, Massachusetts Institute of Technology, Cambridge, USA, 1974.

[Min75] Minsky, M.: Minsky's frame system theory. In: *Proceedings of the 1975 workshop on Theoretical issues in natural language processing*, pp. 104–116. 1975.

[MKA+02] Magkanaraki, A.; Karvounarakis, G.; Anh, T.T.; Christophidesa, V.; Plexousakis, D.: A survey on ontology tools. Technical Report Preprint ST-2000-29243, Project funded by the IST Programme of the Commission of the European, 2002.

[MKC+02] Magkanaraki, A.; Karvounarakis, G.; Christophides, V.; Plexousakis, D.; Anh, T.: Data semantics: what, where and how. Technical Report Preprint TR-CS-02-308, Inst. of CS, Foundation for Research and Technology Hellas, 2002.

[MKSI96] Mena, E.; Kashyap, V.; Sheth, A.; Illarramendi, A.: OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In: *Conference on Cooperative Information Systems*, volume 41:pp. 14–25, 1996.

[Mor38] Morris, C.: *Foundations of the the Theory of Signs*. University of Chicago Press. Chicago, 1938.

[MSBS03] Moody, D.L.; Sindre, G.; Brasethvik, T.; Sølvberg, A.: Evaluating the quality of information models: Empirical testing of a conceptual model quality framework. In: *ICSE*, pp. 295–307. 2003.

[Myl90] Mylopoulos, J.: Object-orientation and knowledge representation. In: *DS-4*, pp. 23–37. 1990.

[Myl98] Mylopoulos, J.: Information modeling in the time of the revolution. In: *Inf. Syst.*, volume 23(3-4):pp. 127–155, 1998.

[NE01] Navathe, S.B.; Elmasri, R.A.: *Fundamentals of Database Systems with Cdrom and Book*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[NF03] Necib, C. Ben; Freytag, J.C.: Ontology based query processing in database management systems. In: *Proceeding on the 6 th international conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems (ODBASE'2003), Catania, Italy*, pp. 37–99. 2003.

[NF04] Necib, C. Ben; Freytag, J.C.: Using ontologies for database query reformulation. In: *Proceeding on the 18 th international Conference on Advances in Databases and Information Systems (ADBIS'2004), Budapest, Hungary*, pp. 173–191. 2004.

[NF05a] Necib, C. Ben; Freytag, J.C.: Query processing using ontologies. In: *Proceedings on the 17th International Conference on Advanced Information Systems Engineering (CAiSE 2005), Porto, Portugal*, pp. 167–186. 2005.

[NF05b] Necib, C. Ben; Freytag, J.C.: Semantic query transformation using ontologies. In: *Proceeding on the 9 th International Database Engineering and Applications Symposium (IDEAS 2005), Montreal, Canada*, pp. 187–199. 2005.

[NFK+00] Nodine, M.H.; Fowler, J.; Ksiezyk, T.; Perry, B.; Taylor, M.C.; Unruh, A.: Active information gathering in Infosleuth. International Journal of Cooperative Information Systems, 2000.

[NH97] Noy, N.F.; Hafner, C.D.: The state of the art in ontology design. In: *AI Magazine*, volume 3(18):pp. 53–74, 1997.

[Noy04] Noy, N.F.: Semantic integration: A survey of ontology-based approaches. In: *SIGMOD Record*, volume 33(4):pp. 65–70, 2004.

[NWP90] Nilakanta, S.; Wemhoff, R.; Prebhu, G. M.: Knowledge-based graph theoretic analysis of data flow diagrams: Integrating CASE tools with expert systems. In: *ACM SIGBDP Conference on Trends and Directions in Expert Systems*, pp. 58–71. 1990.

[Ogb00] Ogbuji, U.: An introduction to RDF. Exploring the standard for Web-based metadata. Web page available at: http://www-128.ibm.com/developerworks/library/w-rdf/, 2000.

[OH01] Orengo, V.M.; Huyck, C.R.: A stemming algorithmm for the portuguese language. In: *SPIRE*, pp. 186–193. 2001.

[OR23] Ogden, C.K.; Richards, I.A.: The meaning of meaning: A study in the influence of language upon thought and of the science of symbolism. In: *London, 10th edition 1969*, pp. 9–12. 1923.

[OV91] Ozsu, M.T.; Valduriez, P.: *Principles of distributed database systems*. Prentice-Hall, Inc., Upper Saddle River, USA, 1991.

[PC02] Pérez, A.G.; Corcho, Ó.: Ontology specification languages for the semantic Web. In: *IEEE Intelligent Systems*, volume 17(1):pp. 54–60, 2002.

[Pea03] Pease, A.: The sigma Ontology development environment. In: *IJCAI-03 Workshop on Ontologies and Distributed Systems (ODS'03), Acapulco, Mexico*, volume 5, pp. 220–238. 2003.

[Pei58] Peirce, C.S.: Collected papers. In: *vols. 1-6, ed. C. Hartshorne and P. Weiss, vols. 7-8, ed., A. W. Burks. Cambridge, Mass.: Harvard Univ. Press*, pp. 22–43. 1931-58.

[PFLC03] Pérez, A.G.; Fernandez-López, M.; Corcho, Ó.: *Ontological Engineering*. Springer Verlag, London Ltd, 2003.

[PLH97] Pirahesh, H.; Leung, T.Y.C.; Hasan, W.: A rule engine for query transformation in Starburst and IBM DB2 C/S DBMS. In: *ICDE'97: Proceedings of the Thirteenth International Conference on Data Engineering*, pp. 391–400. 1997.

[PM01] Pinto, H.S.; Martins, J.P.: A methodology for ontology integration. In: *the First International Conference on Knowledge Capture (K-CAP)*, pp. 368–375. 2001.

[PS05] P. Shvaiko, J. Euzenat: A survey of schema-based matching approaches. In: *Journal on Data Semantics (JoDS)*, volume 2(1):pp. 146–171, 2005.

[PSB+99] Paton, N.W.; Stevens, R.; Baker, P.; Goble, C.A.; Bechhofer, S.; Brass, A.: Query processing in the TAMBIS bioinformatics source integration system. In: *SSDBM*, pp. 138–147. 1999.

[Qua05] Quang, H.L.: Integration of Web Data Sources: A Survey of Existing Problems. In: *17th GI-Workshop über Grundlagen von Datenbanken, Wörlitz*, pp. 78–82. 2005.

[Qui68] Quillian, M.: *Semantic Memory.* Ph.D. thesis, PhD thesis, MIT Press, 1968.

[Ram95] Ram, S.: Intelligent database design using the unifying semantic model. In: *Inf. Manage.*, volume 29(4):pp. 191–206, 1995.

[Rap58] Rapaport, W.J.: Understanding understanding: Syntactic semantics and computational cognition. In: *James E. Tomberlin (ed.), AI, Connectionism, and Philosophical Psychology, Philosophical Perspectives, Atascadero, CA: Ridgeview*, pp. 49–88. 1931-58.

[RB01] Rahm, E.; Bernstein, P.A.: A survey of approaches to automatic schema matching. In: *VLDB J.*, volume 10(4):pp. 334–350, 2001.

[RBG+97] Rector, A.L.; Bechhofer, S.; Goble, C.A.; Horrocks, I.; Nowlan, W.A.; Solomon, W.D.: The GRAIL concept modelling language for medical terminology. In: *Artificial Intelligence in Medicine*, volume 9(2):pp. 139–171, 1997.

[RP00] Rolland, C.; Prakash, N.: From conceptual modelling to requirements engineering. In: *Ann. Software Eng.*, volume 10:pp. 151–176, 2000.

[RS77] Ross, D.T.; Schoman, K.E.: Structured analysis for requirements definition. In: *IEEE Trans. Software Eng.*, volume 3(1):pp. 6–15, 1977.

[RS92] Rusinkiewicz, M.; Sheth, A.P.: Multidatabase applications: Semantic and system issues. In: *VLDB*, p. 545. 1992.

[SAC+79] Selinger, P.G.; Astrahan, M.M.; Chamberlin, D.D.; Lorie, R.A.; Price, T.G.: Access path selection in a relational database management system. In: *SIGMOD Conference*, pp. 23–34. 1979.

[Sag88] Sagiv, Y.: Optimizing datalog programs. In: *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, 1988.

[SBF98] Studer, R.; Benjamins, V.R.; Fensel, D.: Knowledge engineering: Principles and methods. In: *Data Knowledge Engineering*, volume 25(1-2):pp. 161–197, 1998.

[SCM03] Sattler, U.; Calvanese, D.; Molitor, R.: Relationships with other formalisms. In: *Description Logic Handbook*, pp. 137–177. 2003.

[She91] Sheth, A.P.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. In: *VLDB*, p. 489. 1991.

[She95] Sheth, A.: Data semantics: what, where and how. Technical Report Preprint CS-01-99, TR-CS-95-003, LSDIS Lab, Dept. of CS, Univ. of GA, 1995.

[SHKC93] Shekhar, S.; Hamidzadeh, B.; Kohli, A.; Coyle, M.: Learning transformation rules for semantic query optimization: A data-driven approach. In: *IEEE Transactions on Knowledge and Data Engineering*, volume 5(6):pp. 950–964, 1993. ISSN 1041-4347.

[SJB96] Song, W.W.; Johannesson, P.; Bubenko, J.A.: Semantic similarity relations and computation in schema integration. In: *Data Knowledge Engineering*, volume 19(1):pp. 65–97, 1996.

[SJS01] Slivinskas, G.; Jensen, C.S.; Snodgrass, R.T.: Adaptable query optimization and evaluation in temporal middleware. In: *Proceedings of the 2001 ACM SIGMOD international conference on Management of data (SIGMOD '01)*, pp. 127–138. 2001.

[SK93] Sheth, A.P.; Kashyap, V.: So far (schematically) yet so near (semantically). In: *Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*, pp. 283–312. 1993.

[SL97] Sayli, A.; Lowden, B.G.T.: A fast transformation method for semantic query optimization. In: *IDEAS*, pp. 319–326. 1997.

[SM00] Staab, S.; Maedche, A: Axioms are objects, too – Ontology engineering beyond the modeling of concepts and relations. Technical Report 399, TR-399, Institute AIFB, Karlsruhe, 2000.

[Smi87] Smith, B.C.: The correspondence continuum. Technical Report 2, TR CSLI-87-71, Stanford University, 1987.

[SMK97] Steinbrunn, M.; Moerkotte, G.; Kemper, A.: Heuristic and randomized optimization for the join ordering problem. In: *The VLDB Journal*, volume 6(3):pp. 191–208, 1997.

[SO87] Shenoy, S.T.; Ozsoyoglu, Z.M.: A system for semantic query optimization. In: *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pp. 181–195. 1987.

[SO89] Shenoy, S. T.; Ozsoyoglu, Z. M.: Design and implementation of a semantic query optimizer. In: *IEEE Transactions on Knowledge and Data Engineering*, volume 1(3):pp. 344–361, 1989.

[Sow84] Sowa, J. F.: *Conceptual structures: Information processing in mind and machine*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 1984.

[Sow91] Sowa, J. F.: *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann Publication, New York, USA, 1991.

[Sow00] Sowa, J. F.: *Knowledge representation: Logical, philosophical and computational foundations*. Brooks/Cole Publishing Co., Pacific Grove, USA, 2000.

[Sri03] Srivastava, D.: Approximate string joins. In: *15th International Conference on Scientific and Statistical Database Management (SSDBM'03)*, p. 7. 2003.

[SSD92] Shekhar, S.; Srivastava, J.; Dutta, S.: A formal model of trade-off between optimization and execution costs in semantic query optimization. In: *Data Knowl. Eng.*, volume 8(2):pp. 131–151, 1992. ISSN 0169-023X.

[SSKL04] Svab, O.; Svátek, V.; Kavalec, M.; Labský, M.: Querying the RDF: Small case study in the bicycle sale domain. In: *DATESO*, pp. 84–95. 2004.

[SSS92] Siegel, M.; Sciore, E.; Salveter, S.: A method for automatic rule derivation to support semantic query optimization. In: *ACM Trans. Database Syst.*, volume 17(4):pp. 563–600, 1992. ISSN 0362-5915.

[SY94] Sun, W.; Yu, C.T.: Semantic query optimization for tree and chain queries. In: *IEEE Trans. on Data and Knowledge Engineering*, volume 1 (6):pp. 136–151, 1994.

[Tha93] Thalheim, B.: Foundations of entity-relationship modeling. Annals of Mathematics and Artificial Intelligence 7, pages 197-256, 1993.

[UG96] Uschold, M.; Grüninger, M.: Ontologies: principles, methods, and applications. In: *Knowledge Engineering Review*, volume 11(2):pp. 93–155, 1996.

[UHW+98] Uschold, M.; Healy, M.; Williamson, K.; Clark, P.; Woods, S.: Ontology reuse and application. In Proc. of the Int. Conf. on Formal Ontology in Information Systems (FOIS'98), pages 179–192, Amsterdam, 1998.

[Ull88] Ullman, J.D.: *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1988.

[VM96] Vance, B.; Maier, D.: Rapid bushy join-order optimization with cartesian products. In: *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pp. 35–46. 1996.

[WCH+93] Woelk, D.; Cannata, P.; Huhns, M.N.; Shen, W-M; Tomlinson, C.: Using Carnot for enterprise information integration (Synopsis). In: *PDIS*, pp. 133–136. 1993.

[Wie94] Wiederhold, G.: Interoperation, mediation, and ontologies. In: *IEEE Computer*, volume 3:pp. 33–48, 1994.

[Wie98] Wieringa, R.: A survey of structured and object-oriented software specification methods and techniques. In: *ACM Comput. Surv.*, volume 30 (4):pp. 459–527, 1998.

[Woo75] Woods, J.: What's in a link: Foundations for semantic networks. Representation and understanding. In: *Representation and Understanding, D. Bobrow and A. Collins (eds.), Academic Press, New York*, pp. 122–143. 1975.

[WSKR03] Wilkinson, K.; Sayers, C.; Kuno, H.; Reynolds, D.: Efficient RDF storage and retrieval in Jena. In Proceedings of VLDB Workshop on Semantic Web and Databases, pages 131-150, 2003.

[WVV+01] Wache, H.; Voegele, T.; Visser, U.; Stuckenschmidt, H.; Schuster, G.; Neumann, H.; Huebner, S.: Ontology-based integration of information- a survey of existing approaches. In: *17th International Joint Conference on Artificial Intelligence*, pp. 112–139. 2001.

[Xu83] Xu, G.D.: Search control in semantic query optimization. Technical Report TR-83-09, Dept. of Computer Science, Univ. of Massachusetts. Amherst, Massachusetts, 1983.

[YK93] Yoon, J.P.; Kerschberg, L.: Semantic query optimization in deductive object-oriented databases. In: *DOOD*, pp. 169–182. 1993.

[YS89] Yu, C.T.; Sun, W.: Automatic knowledge acquisition and maintenance for semantic query optimization. In: *IEEE Transactions on Knowledge and Data Engineering*, volume 1(3):pp. 362–375, 1989.

[YSP97] Yoon, S.; Song, I.; Park, E.K.: Intensional query processing using data mining approaches. In: *Proceedings of the 6th International Conference on Information and Knowledge Management, Las Vegas, USA*, pp. 201–208. 1997.

[ZAF+03] Zhu, X.; Aref, W.G.; Fan, J.; Catlin, A.C.; Elmagarmid, A.K.: Medical video mining for efficient database indexing, management and access. In: *ICDE*, pp. 569–580. 2003.

[Zha92] Zhang, J.: Classifying approaches to semantic heterogeneity in multidatabase systems. In: *CASCON '92: Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research*, pp. 153–173. 1992.

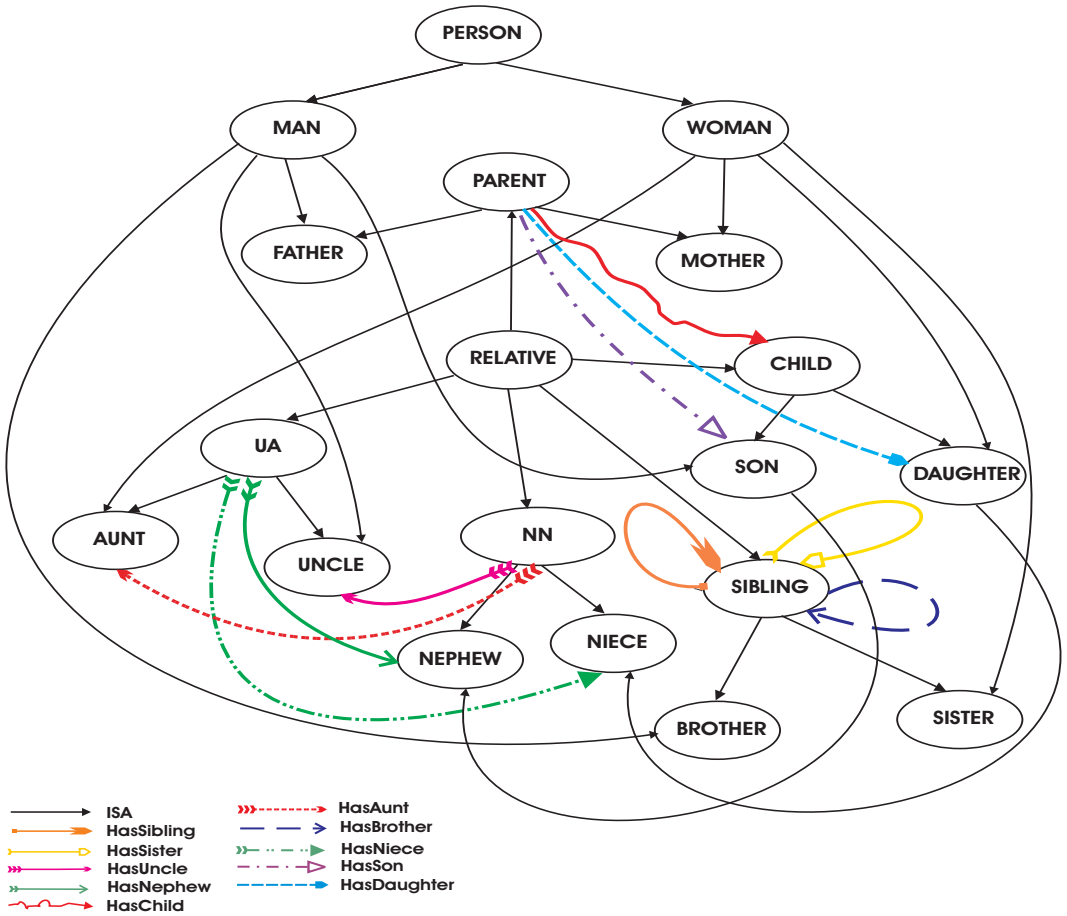
Appendix A

Ontology and Database Examples

This appendix describes the ontology, the database schema, and the mapping information related to the family-example used in Chapter 5. Furthermore, it presents the axioms that define different properties of the ontology relationships.

A.1 Family Example

A.1.1 Family Ontology `FmO`



$$\forall x\ PERSON(x) \Longleftrightarrow MAN(x) \vee WOMAN(x) \quad (A1)$$
$$\forall x\ FATHER(x) \Longleftrightarrow PARENT(x) \wedge MAN(x) \quad (A2)$$
$$\forall x\ MOTHER(x) \Longleftrightarrow PARENT(x) \wedge WOMAN(x) \quad (A3)$$
$$\forall x\ SON(x) \Longleftrightarrow CHILD(x) \wedge MAN(x) \quad (A4)$$
$$\forall x\ DAUGHTER(x) \Longleftrightarrow CHILD(x) \wedge WOMAN(x) \quad (A5)$$
$$\forall x\ BROTHER(x) \Longleftrightarrow SIBLING(x) \wedge MAN(x) \quad (A6)$$
$$\forall x\ SISTER(x) \Longleftrightarrow SIBLING(x) \wedge WOMAN(x) \quad (A7)$$
$$\forall x\ UA(x) \Longleftrightarrow UNCLE(x) \vee AUNT(x) \quad (A8)$$
$$\forall x\ NN(x) \Longleftrightarrow NEPHEW(x) \vee NIECE(x) \quad (A9)$$
$$\forall x\ RELATIVE(x) \Longleftrightarrow UA(x) \vee NN(x) \vee SIBLING(x) \vee CHILD(x) \vee PARENT(x) \quad (A10)$$
$$\forall x\ \exists y\ PARENT(x) \Longrightarrow PERSON(x) \wedge HasChild(x, y) \quad (A11)$$
$$\forall x\ \exists p\ CHILD(x) \Longrightarrow PERSON(x) \wedge HasParent(x, p) \quad (A12)$$
$$\forall x\ \exists y\ SIBLING(x) \Longrightarrow PERSON(x) \wedge HasSibling(x, y) \quad (A13)$$
$$\forall x\ \exists y\ UNCLE(x) \Longrightarrow MAN(x) \wedge (HasNephew(x, y) \vee HasNiece(x, y)) \quad (A14)$$
$$\forall x\ \exists y\ UNCLE(x) \Longrightarrow MAN(x) \wedge HasSibling(x, y) \wedge PARENT(y) \quad (A15)$$
$$\forall x\ \exists y\ AUNT(x) \Longrightarrow WOMAN(x) \wedge (HasNephew(x, y) \vee HasNiece(x, y)) \quad (A16)$$

$\forall x \exists y\ AUNT(x) \Longrightarrow WOMAN(x) \wedge HasSibling(x,y) \wedge PARENT(y)$ $(A17)$
$\forall x\ NIECE(x) \Longleftrightarrow DAUGHTER(x) \wedge (HasUncle(x,y) \vee HasAunt(x,y))$ $(A18)$
$\forall x\ NEPHEW(x) \Longleftrightarrow SON(x) \wedge (HasUncle(x,y) \vee HasAunt(x,y))$ $(A19)$

$\forall x\ y\ HasDaughter(x,y) \Longleftrightarrow HasChild(x,y) \wedge DAUGHTER(y)$ (F1)
$\forall x\ y\ HasSon(x,y) \Longleftrightarrow HasChild(x,y) \wedge SON(y)$ (F2)
$\forall x\ y\ HasSibling(x,y) \Longleftrightarrow HasSibling(y,x)$ (F3)
$\forall x\ y\ z\ HasSibling(x,y) \Longleftrightarrow HasSibling(x,z) \wedge HasSibling(z,y)$ (F4)
$\forall x\ y\ \exists p\ HasSibling(x,y) \Longrightarrow HasParent(x,p) \wedge HasParent(y,p)$ (F5)
$\forall x\ y\ HasParent(x,y) \Longleftrightarrow HasChild^{-1}(x,y)$ (F6)
$\forall x\ y\ HasSister(x,y) \Longleftrightarrow HasSibling(x,y) \wedge SISTER(y)$ (F7)
$\forall x\ y\ HasBrother(x,y) \Longleftrightarrow HasSibling(x,y) \wedge BROTHER(y)$ (F8)
$\forall x\ y \exists p\ HasUncle(x,y) \Longrightarrow HasParent(x,p) \wedge HasBrohter(p,y)$ (F9)
$\forall x\ y \exists p\ HasAunt(x,y) \Longrightarrow HasParent(x,p) \wedge HasSister(p,y)$ (F10)
$\forall x\ y \exists a\ HasNephew(x,y) \Longrightarrow HasSibling(x,a) \wedge HasSon(a,y)$ (F11)
$\forall x\ y \exists a\ HasNiece(x,y) \Longrightarrow HasSibling(x,a) \wedge HasDaughter(a,y)$ (F12)

x, y, w, z, a, p are variables

A.1.2 FmDB-Database Schema

Person(**Pid**, Name, Address, Sex)
PKey(Pid)

Woman(**Wid**)
PKey(Wid)
FKey(Wid) references Person

Man(**Mid**)
PKey(Mid)
FKey(Mid) references Person

Parent(**Eid**)
PKey(Eid)
FKey(Eid) referencess to Person

Parenthood(**Eid**, **Cid**)
PKey(Eid, Cid)
FKey(Eid) references Parent
FKey(Cid) references Child

Father(**Fid**)
PKey(Fid)
FKey(Fid) references Man
FKey(Fid) references Parent

Mother(**Mid**)
PKey(Mid)
FKey(Mid) references Woman
FKey(Mid) references Parent

Son(**Kid**)
PKey(Kid)
FKey(Kid) references Child
FKey(Kid) references Man

Daughter(**Did**)
PKey(Did)
FKey(Did) references Child
FKey(Did) references Woman

Sibling(**Gid**)
PKey(Gid)
Fkey(Gid) references Child

SiblingOf(**Gid1**,**Gid2**)
PKey(Gid1, Gid2)
Fkey(Gid1) references Sibling
Fkey(Gid2) references Sibling

Brother(**Bid**)
PKey(Bid)
FKey(Bid) references Man
FKey(Bid) references Sibling

Child(**Cid**)
PKey(Cid)
FKey(Cid) references Person

Sister(**Sid**)
PKey(Sid)
FKey(Sid) references Woman
FKey(Sid) references Sibling

Uncle(**Uid**)
PKey(Uid)
FKey(Uid) references Man
FKey(Uid) references Sibling

Aunt(**Aid**)
PKey(Aid)
FKey(Aid) references Woman
FKey(Aid) references Sibling

Niece(**Nid**)
PKey(Nid)
FKey(Nid) references Daughter

NieceOf(**Nid**, **UAid**)
PKey(Nid, UAid)
FKey(Nid) references Niece

Nephew(**NEPid**)
PKey(NEPid)
FKey(NEPid) references Son

NephewOf(**NEPid**, **UAid**)
PKey(NEPid, UAid)
FKey(NEPid) references Nephew .

Remarks. FKey(*id-name*) and PKey(*id-name*) denote foreign key and primary key constraints respectively. "FKey(*id-name*) references *Rel-name*": denotes that the key *id-name* references the primary key of the relation *Rel-name*.

A.1.3 Mapping between FmO and FmDB

Relationship	Database Instances
HasChild	$\{(x,y)\|(x,y) \in Parenthood\}$
HasParent	$\{(x,y)\|(x,y) \in Parenthood\}$
HasSibling	$\{(x,y)\|(x,y) \in SiblingOf\}$
HasNephew	$\{(x,y)\|(x,y) \in NephewOf\}$
HasSon	$\{(x,y) \mid (x,y) \in Parenthood \wedge \exists z\ [(z) \in Son \wedge y = z]\}$
HasDaughter	$\{(x,y) \mid (x,y) \in Parenthood \wedge \exists z\ [(z) \in Daughter \wedge y = z]\}$
HasSister	$\{(x,y) \mid (x,y) \in SiblingOf \wedge \exists z\ [(z) \in Sister \wedge y = z]\}$
HasBrother	$\{(x,y) \mid (x,y) \in SiblingOf \wedge \exists z\ [(z) \in Brother \wedge y = z]\}$
HasUncle	$\{(x,y) \mid [(x,y) \in NephewOf \wedge \exists z\ [(z) \in Uncle \wedge y = z]] \vee [(x,y) \in NieceOf \wedge \exists t[(t) \in Uncle \wedge w = t]]\}$
HasAunt	$\{(x,y) \mid [(x,y) \in NephewOf \wedge \exists z\ [(z) \in Aunt \wedge y = z]] \vee [(x,y) \in NieceOf \wedge \exists t[(t) \in Aunt \wedge w = t]]\}$

Table A.1: Mapping between relationships and DB instances: $\Psi_{\Re\mathcal{E}}$

A.2 Relationship Properties

A.2.1 Semantics of generic relationships

In the following we define axioms that describe the semantics of generic relationships. We use predicates Isa, $Synof$, and $Partof$ to represent relationships ISA, SynOF, and PartOf, respectively.

$\forall x\ Isa(x,x)$
$\forall x\ Synof(x,x)$
$\forall x\ Partof(x,x)$
$\forall xyz\ Isa(x,y) \wedge Isa(y,z) \Longrightarrow Isa(x,z)$
$\forall xy\ Synof(x,y) \Longleftrightarrow Synof(y,x)$
$\forall xyz\ Synof(x,y) \wedge Synof(y,z) \Longrightarrow Synof(x,z)$
$\forall xyz\ Isa(x,y) \wedge Synof(y,z) \Longleftrightarrow Isa(x,z)$
$\forall xyz\ Isa(x,z) \wedge Synof(x,y) \Longleftrightarrow Isa(y,z)$
$\forall xyz\ Partof(x,y) \wedge Partof(y,z) \Rightarrow Partof(x,z)$
$\forall xyz\ Partof(x,y) \wedge Synof(y,z) \Longleftrightarrow Partof(x,z)$
$\forall xyz\ Partof(x,y) \wedge Synof(x,z) \Longleftrightarrow Partof(z,y)$

A.2.2 Semantics of domain specific relationships

In the following we define axioms that describe the semantics of some domain specific relationships. We use predicates $Isaspf$, $Madeof$, $Save$, $Usedfor$, $Hascolor$, and $Hastaste$ to represent relationships `ISAspecific`, `MadeOf`, `Save`, `UseFor`, `HasColor`, and `HasTaste`, respectively.

$\forall x\ Isaspf(x,x)$
$\forall x\ Madeof(x,x)$
$\forall xyz\ Save(x,y) \wedge Isa(y,z) \Longrightarrow Save(x,z)$
$\forall xyz\ Save(x,y) \wedge Synof(y,z) \Longrightarrow Save(x,z)$
$\forall xyz\ Madeof(x,y) \wedge Isa(y,z) \Longrightarrow Madeof(x,z)$
$\forall xyz\ Madeof(x,y) \wedge Synof(y,z) \Longrightarrow Madeof(x,z)$
$\forall xyz\ Usedfor(x,y) \wedge Isa(y,z) \Longrightarrow Usedfor(x,z)$
$\forall xyz\ Madeof(x,y) \wedge Usedfor(x,z) \Longrightarrow Usedfor(y,z)$
$\forall xyz\ Synof(x,y) \wedge Hascolor(y,z) \Longrightarrow Hascolor(x,z)$
$\forall xyz\ Synof(x,y) \wedge Hastaste(y,z) \Longrightarrow Hastaste(x,z)$
$\forall xyz\ Isaspf(x,y) \wedge Isaspf(y,z) \Longrightarrow Isaspf(x,z)$
$\forall xyz\ Synof(x,y) \wedge Isaspf(y,z) \Longrightarrow Isaspf(x,z)$
$\forall xyz\ Isaspf(x,y) \wedge Synof(y,z) \Longleftrightarrow Isaspf(x,z)$

Appendix B

Mapping Definitions

This appendix describes the XML-based syntax used for mapping ontologies and databases that are implemented in `ODBT`. It contains

1. Definition of parameters used to connect an ontology with a database.
2. Definition of XML data types for the mappings.

B.1 Definition of Connection Parameters

```
</DbConnection
  jdbcDriver="com.ibm.db2.jcc.DB2Driver"
  jdbcDSN="jdbc:db2://xxx"
  User="xxx”
  Password="xxx" >
</DbConnection>

<!-- ontology connection parameters -->
<OntConnection   ontTool="SESAME">
  <DbConnection
       dbName="MySQL"
        jdbcDriver="com.mysql.jdbc.Driver"
        jdbcDSN="jdbc:mysql://xxx"
        user="xxx"
        Password="xxx" >
</OntConnection>

<Namespace uri="http://localhost/ontologies/product/"></Namespace>
<Namespace uri="http://localhost/ontologies/fruit/"></Namespace>
<Namespace uri="http://localhost/ontologies/bio/"></Namespace>
<Namespace uri="http://localhost/ontologies/entities/"></Namespace>
<Namespace uri="http://www.unine.ch/knowler/wordnet"></Namespace>

<OdbtRelationships>

      <IsARelationship>
      <PrimaryRelationship  url="http://www.w3.org/2000/01/rdfschema#subClassOf" />
      <SecondaryRelationship  url="http://www.unine.ch/knowler/wordnet#hyponymOf" />
      </IsARelationship>

      <PartOfRelationship>
      <PrimaryRelationship  url="http://localhost/ontologies/partWhole/#partOf" />
      <SecondaryRelationship url="http://www.unine.ch/knowler/wordnet#pMeronym" />
      </PartOfRelationship>

      <HasPartRelationship>
      <PrimaryRelationship  url="http://localhost/ontologies/partWhole/#hasPart" />
      </HasPartRelationship>
      <SynonymRelationship>
      <PrimaryRelationship  url="http://www.w3.org/2002/07/owl#equivalentClass" />
      <SecondaryRelationship  url="http://www.unine.ch/knowler/wordnet#simlilarTo" />

       </SynonymRelationship>
       <IsASpecificRelationship>
       <PrimaryRelationship url="http://localhost/ontologies/odbt/#specificSubClass" />
        </IsASpecificRelationship>

</OdbtRelationships>
```

B.2 XML Data-Type Definition for Mappings

```
<!-- Database Ontology Map Document Type Definition -->
<!-- Version 0.1 / 31.7.2005 /  -->
<!ELEMENT DbOntMapping ((DbConnection), (OntConnection),
(DbRelationOntConceptMapping | DbRelationOntRelationshipMapping |
DbAttributeOntRelationshipMapping | DbRelationOntPropertyMapping)*) >
<!ATTLIST DbOntMapping name CDATA #REQUIRED >

<!ELEMENT OntConnection ((DbConnection | XmlFile)?, Namespace+, OdbtRelationships) >
<!ATTLIST OntConnection  ontTool (JENA|SESAME|IBM) #REQUIRED >

<!ELEMENT DbConnection  EMPTY>
<!ATTLIST DbConnection
                dbName (MySQL|PostgreSQL|Oracle|DB2) #REQUIRED
                jdbcDriver CDATA #REQUIRED
                jdbcDSN CDATA #REQUIRED
                user CDATA #IMPLIED
                password CDATA #IMPLIED >

<!ELEMENT XmlFile  EMPTY>
<!ATTLIST XmlFile
                fileName CDATA #REQUIRED>

<!ELEMENT Namespace  EMPTY>
<!ATTLIST Namespace
                uri CDATA #REQUIRED>

<!ELEMENT OdbtRelationships (IsARelationship, PartOfRelationship, HasPartRelationship,
SynonymRelationship , IsASpecificRelationship) >

<!ELEMENT IsARelationship (PrimaryRelationship, (SecondaryRelationship)*) >

<!ELEMENT PartOfRelationship (PrimaryRelationship, (SecondaryRelationship)*) >

<!ELEMENT HasPartRelationship (PrimaryRelationship, (SecondaryRelationship)*) >

<!ELEMENT SynonymRelationship (PrimaryRelationship, (SecondaryRelationship)*) >

<!ELEMENT IsASpecificRelationship (PrimaryRelationship, (SecondaryRelationship)*) >

<!ELEMENT PrimaryRelationship  EMPTY>

<!ATTLIST PrimaryRelationship
                url CDATA #REQUIRED>

<!ELEMENT SecondaryRelationship EMPTY>
<!ATTLIST SecondaryRelationship
                url CDATA #REQUIRED>
```

```
<!ELEMENT DbRelationOntConceptMapping (DbRelation, OntConcept) >

<!ELEMENT DbRelationOntRelationshipMapping (DbRelation, OntRelationship,
(ProjectionFrom, ProjectionTo)?) >

<!ELEMENT DbAttributeOntRelationshipMapping (DbRelation?, OntRelationship,
(ProjectionFrom, ProjectionTo)?) >

<!ELEMENT DbRelationOntPropertyMapping (DbRelation, OntProperty) >

<!ELEMENT ProjectionFrom ((DbRelation) | (DbAttribute, DbRelation)) >

<!ELEMENT ProjectionTo ((DbRelation) | (DbAttribute, DbRelation)) >

<!ELEMENT ForeignKeyTo (DbRelation) >

<!ELEMENT DbRelation (DbAttribute)* >
<!ATTLIST DbRelation
        name CDATA #REQUIRED>

<!ELEMENT DbAttribute ((DbInstance)*,(ForeignKeyTo)?) >
<!ATTLIST DbAttribute
        name CDATA #REQUIRED>

<!ELEMENT DbInstance EMPTY >
<!ATTLIST DbInstance
        name CDATA #REQUIRED>

<!ELEMENT OntConcept EMPTY >
<!ATTLIST OntConcept
        url CDATA #REQUIRED>

<!ELEMENT OntRelationship (OntSubject?, OntObject?) >
<!ATTLIST OntRelationship
        url CDATA #REQUIRED>

<!ELEMENT OntProperty (OntClass)? >
<!ATTLIST OntProperty
        Url CADTA #REQUIRED>

<!ELEMENT OntSubject EMPTY>
<!ATTLIST OntSubject
        url CDATA #REQUIRED>

<!ELEMENT OntObject EMPTY>
<!ATTLIST OntObject
        url CDATA #REQUIRED>

<!ELEMENT OntClass EMPTY>
<!ATTLIST OntClass
        Url CDATA #REQUIRED>
```

Appendix C

UML-Class Diagrams

This appendix presents UML-class diagrams related to the architecture of the prototype `ODBT` including the following components:

- The mapping Interface.
- The ontology Tool Interface.
- The transformation Processor.

For the sake of clarity we omit some classes and methods from the diagrams. Furthermore, we use meaningful names to denote the implemented classes and methods such that their functionalities can be easily interpreted.

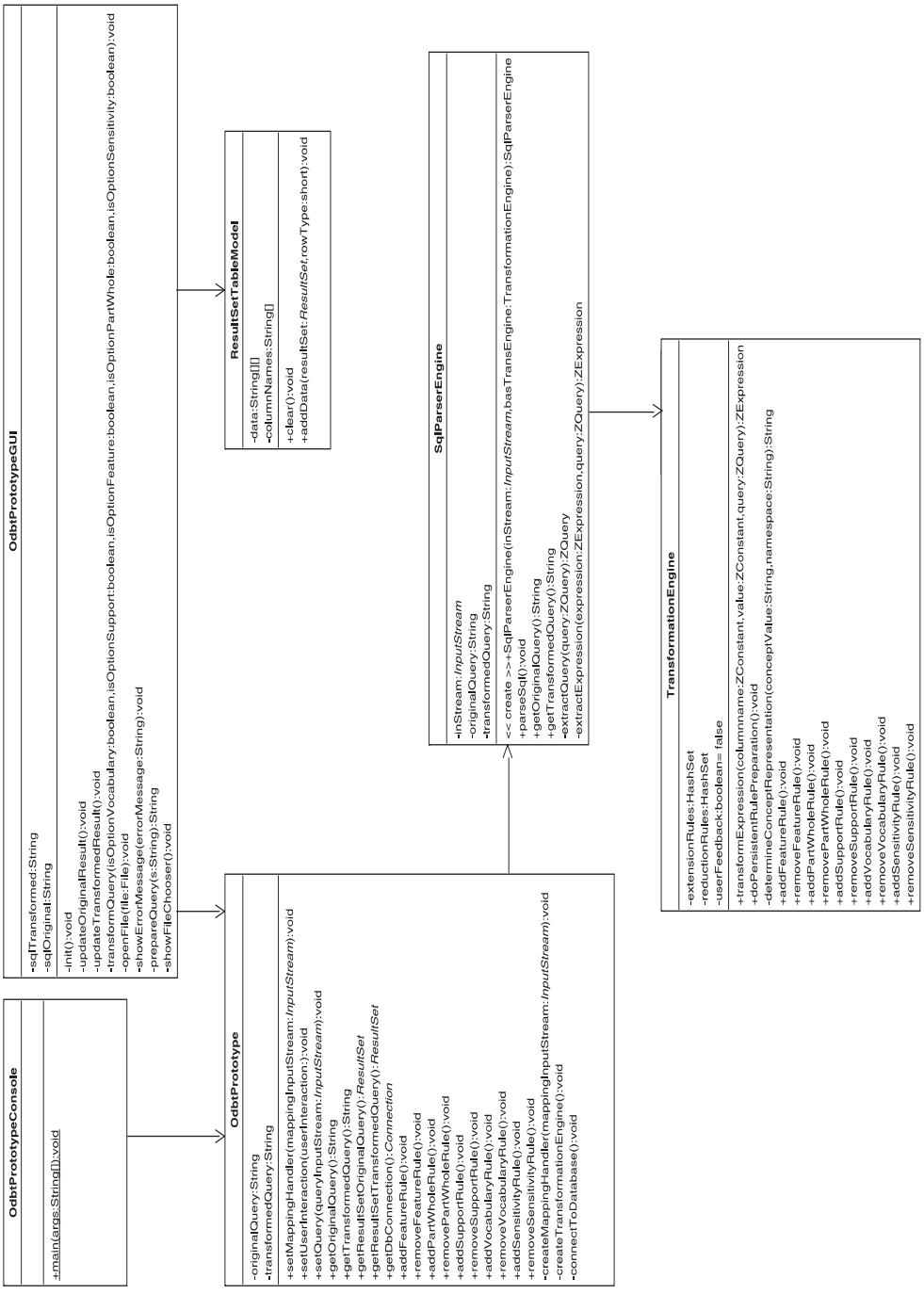


Figure C.1: ODBT-API Implementation

Figure C.2: Mapping Interface Implementation

Figure C.3: Ontology Tool Interface Implementation

Figure C.4: Transformation Processor Implementation

Acknowledgments

Many people have, directly or indirectly contributed to this thesis. Here, I would like to take the opportunity to thank them.

First and foremost, I have much to thank Prof. Johann Christoph Freytag PhD who initiated and supervised my research. His constant support and frequent meetings have made this thesis possible. On a more personal note, his patience and more significantly his perseverance in difficult circumstances were a bright beacon for which I am very grateful. I would also like to thank Professors Felix Naumann and Kai-Uwe Sattler for their helpful comments and feedbacks.

Secondly, I want to thank all the people I worked together with during the last five years. They offered me their friendship, encouragement, ideas, and advice. I am grateful to them all, especially, Dmitri Asonov, Sven Herschel, and Frank Huber who contributed many useful comments to earlier drafts.

Last but not least, my sincere thanks to those close to me, my parents for their loving and dedicated guidance throughout my education, and my friend Tamara for her strong belief in me and her willingness to live several months with a partner under stress.

Berlin, December 11, 2006

Chokri Ben Necib

Curriculum Vitae

Personal information

Chokri Ben Necib
Oldenburger Str. 33
10551 Berlin

Tel.: +49 30 308 247 83
email: necib@dbis.informatik.hu-berlin.de

born 07. 01. 1970 in Grombalia (Tunesia)
nationality: Tunesia
single

Education

1975–1981	Elementary school in Kelibia (Tunesien),
1981–1988	Highschool in Grombalia (Tunesien), Baccalauréat (high school graduation, good)
1988–1990	Training as Computer Technician, Nabeul (Tunesia)
1990–1994	Engineering Study of computer science at the University of Tunis, Tunis, (Tunesia), Diplom Engineer (MS, good)
1994–1995	Learning German at the Studienkolleg of Krefeld (Germany), PNDS Zeugnis (Certificate, good)
1995–1999	Diploma Study of computer science at the RWTH-Aachen University of Technology, Aachen (Germany), Diplom (MS, very good)
2000–2006	Graduate Study of computer science at the Humboldt-University of Berlin, Database group, Berlin (Germany)

Internships

1994	Ministry of foreign affairs of Tunisia, Tunis (Tunisia),
1999	Ericsson Eurolab Deutschland GmbH, Herzogenrath (Germany)
1997–2000	IME-Institute at RWTH-Aachen, Aachen (Deutschland)

Berlin, December 11, 2006

Erklärung

Ich erkläre hiermit, daß

- ich die vorliegende Dissertationsschrift “Ontology-based Semantic Query Processing in Database Systems” selbständig und ohne unerlaubte Hilfe angefertigt habe;
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze;
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist.

11. Dezember 2006